Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Technical University of Cartagena

School of Industrial Engineering

Programming and Commissioning of a Multiple System of 6-axis Anthropomorphic Robots

Bachelor's Final Degree Project

Ginés López Cánovas
Project author

Assoc. Prof. Dr. Óscar de Francisco Ortiz
Assoc. Prof. Dr. Virginijus Baranauskas
Supervisor

Cartagena, Kaunas 2023

Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Technical University of Cartagena

School of Industrial Engineering

Programming and Commissioning of a Multiple System of 6-axis Anthropomorphic Robots

Bachelor’s Final Degree Project

Intelligent Robotics Systems (6121EX013)

Industrial Electronics and Automation Engineering (5071)

Ginés López Cánovas

Project author

Assoc. Prof. Óscar de Francisco Ortiz
Assoc Prof. Dr. Virginijus Baranauskas

Supervisor

Prof. Dr. Vytautas Galvanauskas

Reviewer

Cartagena, Kaunas 2023

Kaunas University of Technology

Faculty of Electrical and Electronics Engineering

Technical University of Cartagena

School of Industrial Engineering

Ginés López Cánovas

Programming and Commissioning of a Multiple System of 6-axis Anthropomorphic Robots

Declaration of Academic Integrity

I confirm the following:

1. I have prepared the final degree project independently and honestly without any violations of the copyrights or other rights of others, following the provisions of the Law on Copyrights and Related Rights of the Republic of Lithuania, the Regulations on the Management and Transfer of Intellectual Property of Kaunas University of Technology (hereinafter – University) and the ethical requirements stipulated by the Code of Academic Ethics of the University;

2. All the data and research results provided in the final degree project are correct and obtained legally; none of the parts of this project are plagiarised from any printed or electronic sources; all the quotations and references provided in the text of the final degree project are indicated in the list of references;

3. I have not paid anyone any monetary funds for the final degree project or the parts thereof unless required by the law;

4. I understand that in the case of any discovery of the fact of dishonesty or violation of any rights of others, the academic penalties will be imposed on me under the procedure applied at the University; I will be expelled from the University and my final degree project can be submitted to the Office of the Ombudsperson for Academic Ethics and Procedures in the examination of a possible violation of academic ethics.

Ginés López Cánovas

Confirmed electronically

APPROVED BY:

KTU Faculty of Electrical and Electronics Engineering

Head of the Department of Automation

Assoc. prof. dr. Gintaras Dervinis

2023 05 04

TASK OF FINAL PROJECT OF UNDERGRADUATE (BACHELOR) STUDIES

Issued to the Student:	*Ginés López Cánovas* Group ERBu 9
1. Project Subject:	
Lithuanian Language:	Daugybinės 6-ašių antropomorfinių robotų sistemos programavimas ir paleidimas
English Language:	Programming and Commissioning of Multiple System of 6-axis Anthropomorphic Robots

Approved 2023 May 3d. Decree of Dean Nr. *V25-03-6*

2. Goal of the Project:	Programming and commissioning of a dual robotic station, made up of two anthropomorphic arms and its own GUI
3. Specification of Final Project:	The work must meet the methodological requirements for the preparation of final projects of KTU Faculty of Electricity and Electronics.

4. Project's Structure. *The content is concretized together with supervisor, considering the format of the final project, which is listed in 14 and 15 points of Combined Description of Preparation , Defence and Keeping of Final Projects Methodical Requirements*

4.1 Analyse structure, assembly and programming of robotic arm

4.2 Choose technological components for robotic arm and asseble them

4.3 Experimental testing of the arms

4.4 Create graphical interface for controlling of the arms

5. Economical Part. *If economical substantiation is needed; content and scope is concretized together with supervisor during preparation of final projects*

none

6. Graphic Part. *If necessary, the following schemes, algorithms and assembly drawings; content and scope is concretized together with supervisor during preparation of final projects*

Show component assembling schemes, graphical interface and explain them

7. This Task is Integral Part of Final Project of Undergraduate (Bachelor) Studies

8. The Term of Final Project Submission to Defense Work at a Public Session of Qualification Commission. *until 2023-06-01*
(date)

I received this task:	*Ginés López Cánovas*	*2023-02-08*
	(student's name, surname, signature)	*(date)*
Supervisors:	**Assoc. Prof. Dr. Virginijus Baranauskas** **Assoc. Prof. Dr. Óscar de Francisco Ortiz**	*2023-02-08*
	(position, name, surname, signature)	*(date)*

López Cánovas, Ginés. Programming and commissioning of a multiple system of 6-axis anthropomorphic robots. Bachelor's Final Degree Project / supervisors: Assoc. Prof. Dr. Óscar de Francisco Ortiz, Assoc. Prof. Dr. Virginijus Baranauskas; School of Industrial Engineering; Technical University of Cartagena; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): electronics engineering (engineering science).

Keywords: robotics, programming, graphical interface, open source robotic arms, robot coordination, robotic dual station.

Kaunas, Cartagena, 2023. 71 pages.

Summary

The objective of the final project is the programming and commissioning of a dual robotic station, made up of two anthropomorphic arms and its own GUI. This project includes: an introduction, where the objective and reasons for the work are presented; a theoretical analysis, where the necessary knowledge about robotics, microcontrollers, programming and open source robotic arms is provided to carry out the work; a theoretical section, where the choice of components is made, the design of the electrical and control system, and the structure and flowchart of the robotic station and its microcontrollers are proposed. The experimental section develops the points seen in the theoretical section, showing the assembly of the robotic station, the programming of each of its components and the implementation of the wireless screen. The conclusions and a list of bibliographical references are presented.

López Cánovas, Ginés. Programación y puesta en marcha de sistema múltiple de robots antropomórficos de 6 ejes. Trabajo de Fin de Grado / supervisores: Assoc. Prof. Dr. Óscar de Francisco Ortiz, Assoc. Prof. Dr. Virginijus Baranauskas; Universidad Politécnica de Cartagena, Facultad de Ingeniería Industrial; Universidad Tecnológica de Kaunas, Facultad de Ingeniería Eléctrica y Electrónica.

Campo de estudio y área: ingeniería electrónica (ciencias de la ingeniería).

Keywords: robótica, programación, interfaz gráfica, brazos robóticos de código abierto, coordinación de robots, estación robótica dual.

Kaunas, Cartagena 2023. 71 páginas.

Resumen

El objetivo del proyecto final es la programación y puesta en marcha de una estación robótica dual, formada por dos brazos antropomórficos y su propia GUI. Este proyecto incluye: una introducción, con la que se presenta el objetivo y motivos del trabajo; un análisis teórico, con el que se aportan los conocimientos necesarios sobre robótica, microcontroladores, programación y brazos robóticos de código abierto para realizar el trabajo; una sección teórica, en la que se realiza la elección de componentes, el diseño del sistema eléctrico y de control, y se plantea la estructura y diagrama de flujo de la estación robótica y sus microcontroladores. La sección experimental desarrolla los puntos vistos en la sección teórica, mostrando el ensamblaje de la estación robótica, la programación de cada uno de sus componentes y la implementación de la pantalla inalámbrica. Se presentan las conclusiones y una lista de referencias bibliográficas.

López Cánovas, Ginés. Daugybinės 6-ašių antropomorfinių robotų sistemos programavimas ir paleidimas. Bakalauro baigiamasis projektas / vadovai / doc. dr. VIrginijus Baranauskas, doc. dr. Óscar de Francisco Ortiz; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas, Universidad Politécnica de Cartagena, Pramonės inžinerijos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, inžinerijos mokslai.

Reikšminiai žodžiai: robotika, programavimas, grafinė sąsaja, atvirojo kodo robotinės rankos, robotų koordinavimas, robotinė dviguba stotis.

Kaunas, Cartagena 2023. p. 71

Santrauka

Baigiamojo projekto tikslas – suprogramuoti ir paleisti dvigubą robotų stotį, sudarytą iš dviejų antropomorfinių rankų ir savo GUI. Šį projektą sudaro: įvadas, kuriame pristatomas darbo tikslas ir priežastys; teorinė analizė, kur apibūdinamos darbui atlikti reikalingos žinios apie robotiką, mikrovaldiklius, programavimą ir atvirojo kodo robotines rankas; teorinė dalis, kurioje pasirenkami komponentai, siūloma elektros ir valdymo sistemos konstrukcija, robotizuotos stoties ir jos mikrovaldiklių struktūra ir schema. Eksperimentinėje dalyje pristatomi teorinėje dalyje aptarti aspektai, parodomas robotų stoties surinkimas, kiekvieno jos komponento programavimas bei belaidžio ekrano sudarymas. Pateikiamos išvados ir bibliografinių nuorodų sąrašas.

Table of contents

List of figures

List of tables

List of abbreviations and terms

Abbreviations:

GUI	Graphic User Interface
DOF	Degrees Of Freedom
UART	Universal Asynchronous Receiver/Transmitter
ROS	Robotic Operative System
GPIO	General Purpose Input/Output

Introduction

In the field of robotic arms, there have been constant advancements since their inception, enabling a growing variety of processes to be automated year after year. However, there were numerous tasks that could not be automated using a single robotic arm, but instead required human intervention. This led to the development of a new type of robot, the dual arm robot, consisting of two arms capable of working together and coordinating their actions.

To automate these types of tasks, research has been focused on dual arm robots, which consist of two robotic arms that require more precise and advanced control to work together on the same task. These robots offer many new possibilities, including the ability to automate tasks that were previously only possible for humans to perform, such as manipulation or complex assembly of parts and even surgeries. Before the advent of dual arm robots, these tasks could only be performed by highly trained individuals, with all the limitations associated with humans: fatigue, physical constraints, efficiency, and precision.

Moreover, developing a low-cost model of dual arm robots would be useful for assisting people with physical disabilities or difficulties, enabling them to perform certain activities or tasks that they otherwise could not do. Therefore, the objective of this project is to develop a functional dual robotic station, consisting of two anthropomorphic robotic arms, each being an existing open-source model, low-cost and 3D printed.

In addition, this robotic station will be prepared with a central GUI, allowing both arms to be controlled in real-time and to create, save, and execute coordinated paths between them without the need of an external computer. This will create a functional dual arm robotic station for academic and research purposes, enabling future developments in this field.

Objectives

The objective of the final project is the programming and commissioning of a dual robotic station, made up of two anthropomorphic arms and its own GUI.

To achieve this goal, it is divided into the following sub-objectives:
1. Research on open source robotic arm models and selection.
2. Selection and design of necessary electrical and electronic components for the robotic station.
3. Robots assembly.
4. Robotics Station Integration of both arms with the central box.
5. Design and programming of the closed-loop motion control system.
6. Implementation of the wireless touch panel to the central computer.
7. Design and programming of the app responsible for coordinating the robots and remote control.

1. Theoretical Annalysis

1.1. Robotic Arms

Robotic arms first emerged in the 1960s and were initially used in various industrial processes such as welding or assembly of production parts [1]. The first robotic arm was Unimate (see **Fig. 1**).

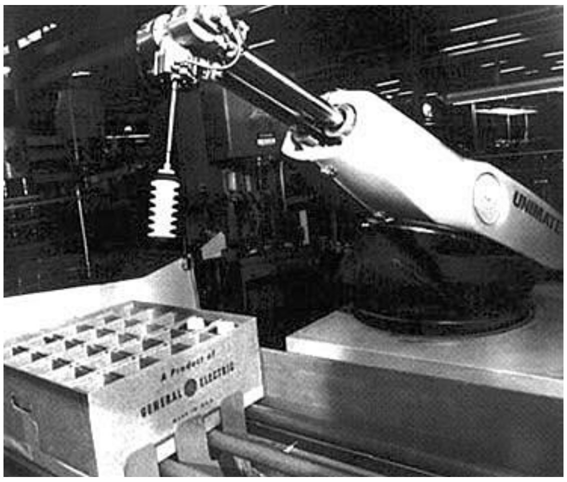


Fig. 1. Unimate, the first robotic arm [2]

Technological advancements and demand from various industries have led to the development of different types of robots, including Cartesian robots, SCARA robots, collaborative robots, and articulated robots. Articulated robots are those with multiple joints, allowing them to perform complex movements. Within this type of robot, are the so-called anthropomorphic arms.

1.1.1. Antropomorphic Robotic Arm

An anthropomorphic robotic arm is a type of robotic arm with a structure similar to a human arm (see **Fig. 2**), allowing them to perform complex movements and operations.

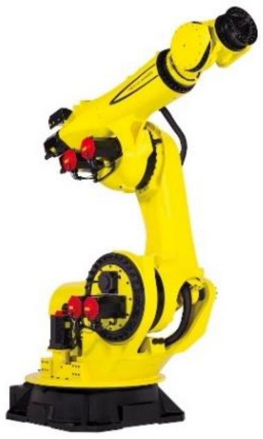


Fig. 2. Generic antropomorphic robotic arm [3]

Their structure is formed by a series of joints that simulate the anatomy of a human arm: shoulder, elbow, and wrist. Therefore, these robots are articulated arms formed by at least six DOF, because with less joints it can‘t simulate all the movements of a human arm [4]. However, some arm models have a greater number of DOF, allowing for greater movement capacity.

1.1.2. Dual Arm Robot

Dual arm robots are an evolution of conventional industrial robots, driven by the need to automate increasingly complex tasks and improve the versatility and capabilities of robots. Dual arm robots are formed by two robotic arms (see **Fig. 3**) which enable them to perform more complex tasks that require greater maneuverability and were previously more difficult to automate or had to be performed by humans[5].

Fig. 3. YuMi, the ABB dual arm robot [6]

This type of robot requires very precise control, as each arm must take into account the position of the other arm in order to perform a joint operation. If the position of the end-effector of one arm is not exactly what the other arm expects, this can cause the arms to not work together and the task cannot be completed[7].

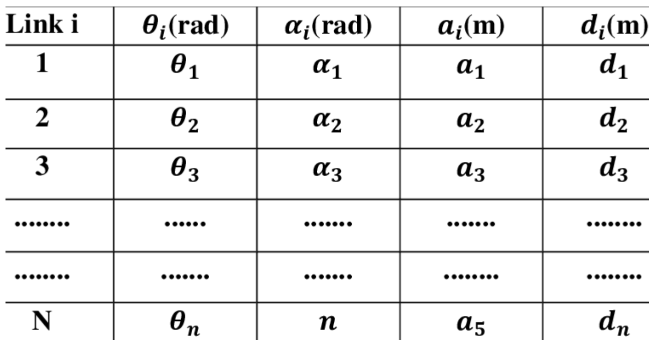
1.2. Mathematical model of a robotic arm

Creating a mathematical model of a robot is crucial as it allows us to understand the geometry and behavior of the robot. This model can also enable the subsequent calculation of the robot's kinematics and dynamics.

One of the most commonly used methods for modeling a robot is to obtain the Denavit-Hartenberg parameters for each of the robot's joints [8]. These parameters are as follows:

- Link length (d): The perpendicular distance from the Z_{i-1} axis to the Z_i axis, measured along the X_i axis.
- Rotation angle around the previous Z-axis (θ): The angle between the Z_{i-1} axis and the Z_i axis, measured around the X_{i-1} axis.
- Link offset distance (a): The perpendicular distance from the X_{i-1} axis to the X_i axis, measured along the Z_i axis.
- Rotation angle around the current X-axis (α): The angle between the X_{i-1} axis and the X_i axis, measured around the Z_i axis.

By obtaining these parameters, a table can be created, known as the D-H table (see **Fig. 4**), which describes the robot's geometry. These parameters can be used later for calculating the robot's forward kinematics and inverse kinematics.



Link i	θ_i(rad)	α_i(rad)	a_i(m)	d_i(m)
1	θ_1	α_1	a_1	d_1
2	θ_2	α_2	a_2	d_2
3	θ_3	α_3	a_3	d_3
....	......	....	....	....
....	....	....	....	....
N	θ_n	n	a_5	d_n

Fig. 4. Denavit-Hartenberg table [9]

1.2.1. Forward kinematics

The forward kinematics is a mathematical calculation that enables us to determine the position and orientation of each component of a robot, based on its mathematical model and the angles of its individual joints. If the D-H table of the robot has been obtained, the Denavit-Hatenberg algorithm can be used to obtain the forward kinematics.

1.2.2. Inverse kinematics

Unlike forward kinematics, inverse kinematics can calculate the configuration of a robot (angles at which each joint should be positioned) in order for the end-effector or a specific joint to be in a certain position and orientation.

However, calculating the inverse kinematics for arms with more than three degrees of freedom (DOF) can become very complicated, as opposed to forward kinematics, since this calculation can have multiple possible results for each position and orientation.

To calculate inverse kinematics, different methods can be used, such as analytical methods and iterative methods. Analytical methods use mathematical equations to obtain exact solutions, but they can only be used for robots with simple geometries, such as a two or three DOF robot, as the complexity of the equations and the number of possible results increase significantly for more complex geometries. Iterative methods, on the other hand, use an iterative approach to reach a solution, meaning that they adjust the joint angle values in each iteration until the result is close enough to a valid configuration. These methods can be used for robots with complex configurations.

Therefore, to calculate the inverse kinematics of a 6-DOF anthropomorphic arm can be used an iterative method [10] such as Inverse Jacobian, Pseudo-inverse, Damped least squares, Newton-Raphson or Levenberg-Marquardt [11].

One of the main issues when calculating the inverse kinematics of a robot is the presence of singularities, which are configurations in which the robot loses one or more degrees of freedom, and the calculation of the inverse kinematics becomes indeterminate or impossible.

To solve this problem, the Levenberg-Marquardt method has proven to be very useful, avoiding singularities and providing stable solutions in the presence of singular configurations in robots with multiple degrees of freedom [12].

1.3. Control Systems of a robotic arm

Robotic arms are composed of different motors, sensors, and actuators. In this way, a robotic arm moves in a certain space to perform certain tasks. To achieve this, different types of control can be applied, which will determine its movement and how it executes these tasks.

Two types of classification of these control systems can be made:
- Classification based on the controlled physical magnitude.
- Classification based on the type of control loop.

1.3.1. Physical magnitude-based classification

The control of different physical magnitudes can be classified as:
- Position
- Speed
- Acceleration
- Torque

1.3.2. Looptype-based classification

Open-loop control (**Fig. 5**): This type of control moves the robotic arm motors based on the expected operation, following a pre-defined programming, without considering any external disturbances or unexpected events that may occur, such as a motor skipping a step or colliding with an obstacle, since it does not have sensors to measure these physical magnitudes.

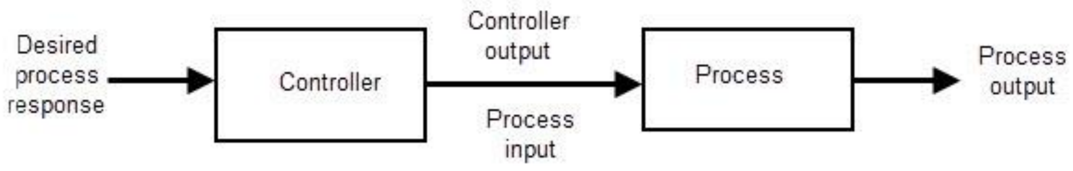


Fig. 5. Open-loop control schema [13]

Close-loop control (**Fig. 6**): In this case, sensors or encoders are used to measure real-time data from the system. Therefore, the physical properties of the system in real-time can be measured, adjusting the control action of the system.

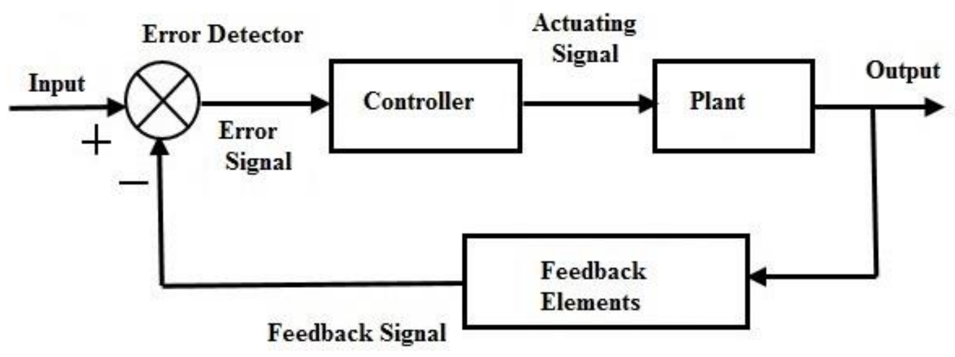


Fig. 6. Close-loop control schema [14]

1.4. Serial communication protocols

Serial communication protocols are commonly used for short-distance communication, and microcontrollers typically support two types of serial communication protocols:

1.4.1. I2C

I2C is a synchronous serial communication protocol. It can handle multiple devices in the same connection. It uses two lines for the communication, SDA (Serial Data Line) and SCL (Serial Clock Line). Its architecture is master-slave, so, in an I2C connection the master will start a connection and send information, which will be received by the slaves.

1.4.2. UART

UART is an asynchronous serial communication protocol. It can connect two devices, and both of them can send and receive information in the same UART connection. It uses two lines for the communication, Tx(to transmit information) and Rx(to receive information). The Tx pin of each device is connected to the Rx pin of the other device.

1.5. Programming tools for robotics

In the field of robotics, different tools and libraries have been developed, making the programming of complex robotic systems easier and more effective. In fact, the use of these tools can be one of the main factors in deciding which language to program in or which hardware to select to control the robot.

1.5.1. Marlin

Marlin is an open-source firmware written in C++, and it is widely used for controlling 3D printers, CNCs, and other machines based on a microcontroller. Due to its high configurability and efficiency, it is also popularly used in open-source robotic arms[15].

Once installed on the microcontroller, this firmware can receive commands in the standard "Gcode" by a serial port. With these commands, the firmware can move the motors, change the speed, acceleration, actuate the gripper, among other functionalities.

1.5.2. ROS

ROS is an open-source framework widely used in robotics, as it provides tools and solutions for most robotics projects and it is compatible with several programming languages, including C++ and Python.

It is highly modular, and one of its characteristics is that these modules can even be distributed across different physical devices and connected through the network, making it easier to create complex robotic systems for all types of tasks. Furthermore, ROS has a wide variety of packages developed for all kinds of robotic applications, from moving a robotic arm to autonomous navigation, allowing them to be reused for other projects.

For the control of a robotic arm, the package MoveIt! is available, which allows the control and motion planning of a robotic arm. It provides a pre-built graphical interface with a 3D environment to visualize the robot's movements in real-time[16].

MoveIt! is a very useful package, but it is computationally heavy to run, so it would be necessary to run it on an external computer, making impossible to use it in an embbeded system based in a microcontroller or a single-board compuer, because the GUI and the simulations are too heavy to run.

1.5.3. Accelstepper library

The AccelStepper library is an Arduino library used to control stepper motors, allowing to control the speed and the acceleration in each movement. Additionally, this library allows us to control multiple stepper motors, so all the motors grouped in a multiple controller start and end their movement synchronously.

This makes this library a very useful tool for a precise control of robots with stepper motors based on a microcontroller such as Arduino, as it is optimized to realized coordinated control of multiple motors despite Arduino isn‘t able to use multi-threading.

1.5.4. Robotics Toolbox for Python

This is an open-source Python library that provides functions for mathematical analysis and simulation of robotic systems. This library is based on the famous MATLAB toolbox for robotics and was developed by Peter Corke, a robotics professor at the University of Queensland[17].

This library can be used to create the mathematical model of a robot and realize operations with it, such as direct and inverse kinematics calculation, dynamics, collisions, and drawing the robot in a 3D plot, among other things.

In general, it provides tools for working with different types of robots, such as mobile robots, industrial robots, and manipulator robots, and has many tools for visualizing and simulating robotic systems.

1.6. Programming languages for GUI creation

1.6.1. C++

C++ is a general-purpose programming language that was developed in 1983. It is an extension of the C language, implementing object-oriented features. C++ is a compiled programming language with low runtime overhead, and it is used in a wide variety of applications. One of these applications is the creation of GUI, which can be done easily through libraries such as Qt[18].

Qt is an open-source framework used for creating GUI. This framework is easy to use and can develop cross-platform applications without having to rewrite the code enterely for each operating system.

1.6.2. Python

Python is an interpreted, high-level programming language designed to be very easy to write and read. Being interpreted, it does not need to be compiled, as the machine interprets the code and transforms it into machine code in real-time during execution. Additionally, this language includes several different libraries for creating GUIs, including PyQt and Tkinter[19].

PyQt is a library that allows for creating graphical applications using the Qt framework mentioned before. PyQt allows for creating user interfaces utilizing the power and versatility of Qt, while also taking advantage of the simplicity and ease of use of Python. This library is the best option for creating a complex project with an advanced GUI.

On the other hand, Tkinter is the default Python library for creating graphical user interfaces. Tkinter is a very simple and easy-to-use library that is distributed along with most Python installations and uses the Tk graphics toolkit. It is highly customizable and ideal for creating simple and elegant interfaces, as it includes a large amount of functionality and is integrated into Python, making it very easy to get started with.

1.7. Development boards

There are different types of development boards available nowadays, which can be used to create an embedded system[20]. These boards can be mainly divided into two main types:

1.7.1. Microcontroller boards

This section is about boards designed around a microcontroller chip, which is a small computer that includes its own processor, memory, and input/output peripherals on a single integrated circuit. Microcontrollers have limited computing power, so they are usually programmed in C/C++.

However, there are other tools that allow writing code for microcontrollers in other languages such as Python or Javascript. These microcontrollers can be programmed, and the code they contain will be executed when the board is turned on. There are different types of microcontroller boards, such as:

– Arduino: It is the most famous development board in the world, and it has different models depending on the needs. The main models are Arduino Nano, Arduino Uno, and Arduino Mega.

- ESP: These development boards have become popular in recent years and have the peculiarity that, unlike Arduino boards, they have integrated Bluetooth and Wifi connections. The main models are ESP32 and ESP8266.
- Raspberry: Despite mainly creating single-board computers, they also have a microcontroller board: Raspberry Pi Pico.

A comparision between main microcontroller-boards is shown in **Table 1**.

Table 1. Microcontroller boards comparision [21]

Board	**GPIO**	**UART**	**I2C**	**Precessor**	**Memory**	**Wifi**	**Bluetooth**
Arduino Mega	54	4	2	ATmega2560	256 KB Flash	No	No
Arduino Uno	20	1	1	ATmega328P	32 KB Flash	No	No
Arduino Nano	22	1	1	ATmega328P	32 KB Flash	No	No
ESP32	36	3	2	ESP32-D0WDQ6	520 KB SRAM	Yes	Yes
Raspberry Pi Pico	26	1	1	RP2040 Microcontroller	264 KB SRAM	No	No

1.7.2. Single-board computer

These boards are designed to function as credit-card-sized computers that are capable of running a complete operating system. The most well-known brand is Raspberry, which has progressively developed increasingly powerful models, culminating in the current Raspberry Pi 4.

On these boards can be insatalled a complete operating system such as Linux or Android. Its official operating system is called Raspberry Pi OS, which is based on the Debian distribution of Linux.

Because of it in these boards you can use any programming language to create any type of program, as they function like a regular computer. In this way, these boards have GPIO outputs like microcontroller boards, while also possessing basic computer ports such as USB ports, HDMI, audio jack, and Ethernet port.
A comparision between main Raspberry Pi models is shown in **Table 2**.

Table 2. Raspberry Pi versions comparision [22]

Board	**Raspberry Pi 3**	**Raspberry Pi 3B+**	**Raspberry Pi 4**
Processor	Broadcom BCM2837B0 64-bit quad-core ARM Cortex-A53	Broadcom BCM2837 64-bit quad-core ARM Cortex-A53	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
CPU clock speed	1.2 GHz	1.4 GHz	1.5 GHz
RAM	1 GB	1 GB	1 GB, 2 GB, 4 GB or 8 GB
Ethernet	10/100 Mbps	Gigabit	Gigabit
Wireless	802.11n	802.11n/ac	802.11ac
Bluetooth	4.1	4.2	5.0
USB ports	4	4	2x USB 2.0, 2x USB 3.0
HDMI	1	1	2 (micro HDMI)
GPIO pins	40	40	40

1.8. Open-Source Robotic Arms

Currently, there is a wide variety of open-source robotic arm projects, ranging from simple robots (3DoF) consisting of 3 servomotors[23], to much more complex robots such as the BCN3D Moveo, which has 6 DoF, uses stepper motors to move its joints, and employs a modified version of Marlin firmware, which is an open-source firmware commonly used in 3D printers and CNC machines and controlled through Gcode.

In this section, I will present the characteristics, advantages, and disadvantages of different types of open-source robotic arms with at least 6 DoF and controlled by stepper motors. The models that will be presented are shown in **Table 3**.

Table 3. Open-Source robotic arms comparision

Robot	**DoF**	**Payload (Kg)**	**Reach (mm)**	**Cost (€)**	**Firmware**	**Feedback**
BCN3D Moveo	6	0.8	538	400	Marlin	No
SmallRobotArm	6	¿?	225	¿?	Custom	No
Thor	6	0.75	422.15	379	Marlin	No
Thor with addons	6	0.75	598	500	Custom	Yes

1.8.1. BCN3D Moveo

This is a robotic arm developed by the company BCN3D Technologies, a Spanish company dedicated to the development of 3D printing. The 6-DOF arm (see **Fig. 7**) is 3D printed and uses an Arduino Mega, whose firmware is a modified Marlin. Therefore, to control it, GCode must be sent from a computer, making use of a generic GCode sender.

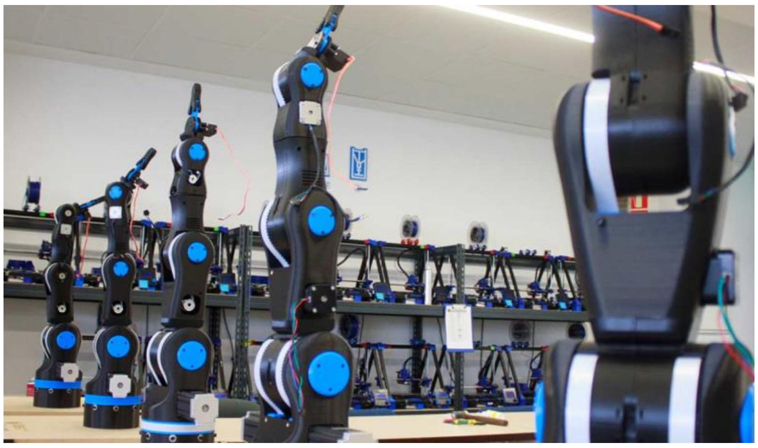


Fig. 7. BNC3D Moveo robotic arm [24]

Advantages:

- In comparison with other models, it has a great reach of 538mm.
- Low cost.
- It is mounted on a highly optimized and robust firmware.

Disadvantages:

- It can only be controlled by direct kinematics. To control it by inverse kinematics, an inverse kinematics calculator must be used in a separate software.
- It does not include sensors for closed-loop control.

1.8.2. SmallRobotArm

SmallRobotArm is a 6 DOF robotic arm of small dimensions (see **Fig. 8**) that uses its own Arduino code to control the position of the joints. The arm has smaller dimensions compared to the other selected robotic arms, and it is not completely 3D-printed, as it has several metallic parts to give it structural integrity.

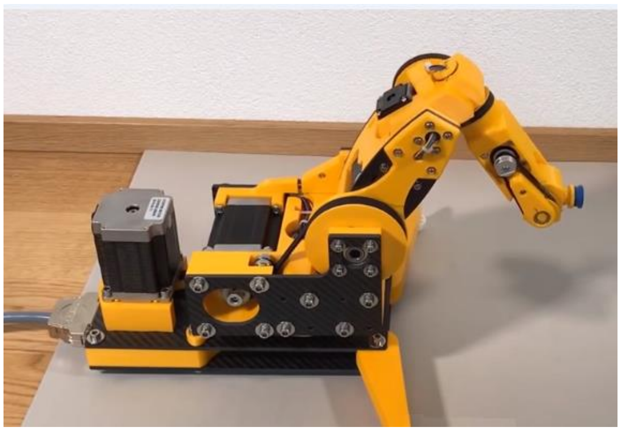


Fig. 8. SmallRobotArm robotic arm [25]

Advantages:

- Its firmware incorporates the calculation of direct and inverse kinematics.

Disadvantages:

- Low reach
- Low payload
- Most of the arm is not 3D-printed, so it is not easily modifiable.
- It does not incorporate feedback in its joints.
- Its firmware does not include a communication protocol.

1.8.3. Thor

This is a 6DOF arm, fully 3D printed (see **Fig. 9**) and controlled by an Arduino Mega [26]. Like the BCN3D model, it uses a modified Marlin firmware. However, unlike the BCN3D robot, this arm has its own software for sending GCode from a computer, called "Asgard". Despite this, the software is only capable of performing control in direct kinematics, and can only send movement commands to a robot.

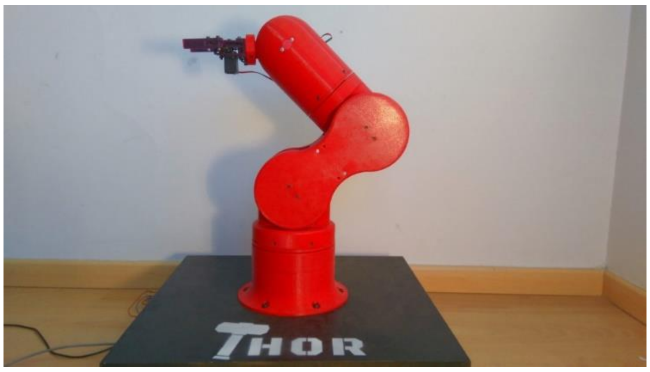


Fig. 9. Thor robotic arm [27]

Regarding its electronics, this robot has its own custom PCB, which once assembled can be connected to the Arduino Mega to control all the stepper motors of the robot. This board was created because existing shields for an Arduino Mega only support 5 stepper motors, while this PCB supports up to 7 stepper motors.

Advantages:

- Great reach of 422.15mm
- Low cost
- Optimized and robust firmware (Marlin)
- Includes its own PC software for sending GCode to the robot.

Disadvantages:

- Can only be controlled in direct kinematics; inverse kinematics requires a separate kinematics calculator.
- Does not include sensors for closed-loop control.
- Its software only works for one arm, and can only send commands in direct kinematics.

1.8.4. Thor with addons

This robotic arm (see **Fig. 10**), developed by dannyvendeheuvel, is a modification of the previously seen Thor model.

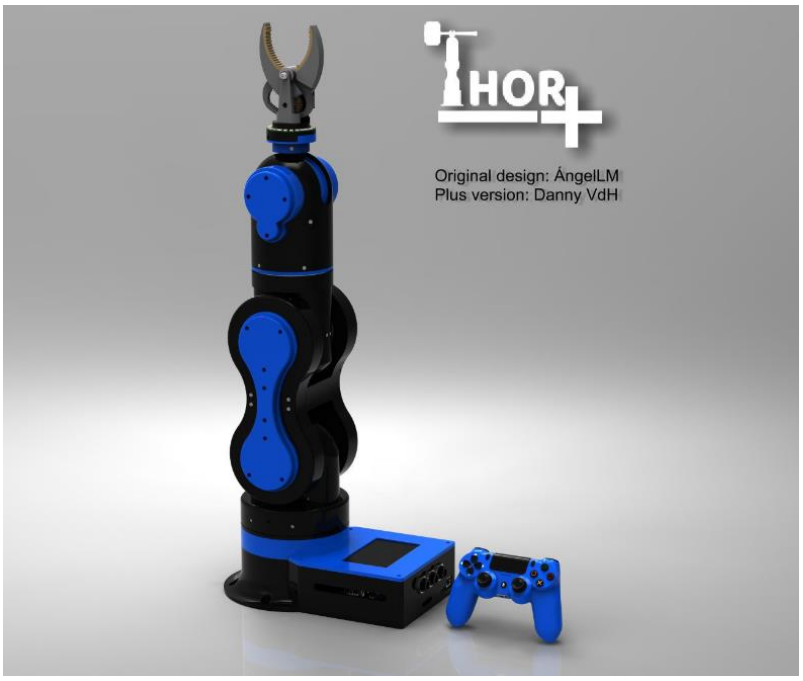


Fig. 10. Thor with addons wobotic arm [28]

This model also includes significant modifications from the original robot, such as:

- Modified STL files, so that encoders can now be placed on each joint, allowing for closed-loop motion control.
- Change of microcontroller from an Arduino Mega to an Ultratonics Pro v1.0 board, which is a 32-bit board that supports multi-threading and is commonly used for 3D printing.

Advantages:

- Implements the possibility of closed-loop position control.
- Greater reach than the original Thor model.
- Improved microcontroller used.

Disadvantages:

- No firmware or software developed.
- The improved microcontroller results in a considerable increase in budget.

2. Theoretical approach

2.1. Robotic arm selection

The first step consists of choosing which robotic arm will be used to develop the dual robotic station. Among the options of open-source anthropomorphic arms mentioned previously, the arms with the greatest range of motion are Thor and BCN3D Moveo.

Between these two options, the version of Thor with addons stands out, because it already incorporates in the design support for encoders on each joint. This model is a robotic arm with feedback in each of its axes, with an intermediate microcontroller that will handle the gripper control and sensor data collection, so that only two signal cables have to run through the arm, in addition to the power supply and motor cables.

But the major disadvantage of this model is that having feedback requires the use of a more powerful and substantially more expensive microcontroller, in this case the Ultratonics v1.0 board, which supports multi-threading and provides the possibility of controlling the motors while checking the position of each joint, but it has a price ten times higher than an Arduino mega, which is the controller used in the other models.

Despite this, and due to budget limitations, it was decided to use an Arduino Mega, which has sufficient power for the simultaneous control of 7 stepper motors, performing feedback each time the motors reach the setpoint. To be able to do this, a new firmware had to be developed from scratch for being efficient.

Therefore, the Thor with addons model will be used for the robotic station, with an Arduino Mega as the main controller and an Arduino nano for sensor reading and gripper control.

2.2. Mathematical model of the robot

This robotic arm is divided into 6 joints as shown in **Fig. 11**.

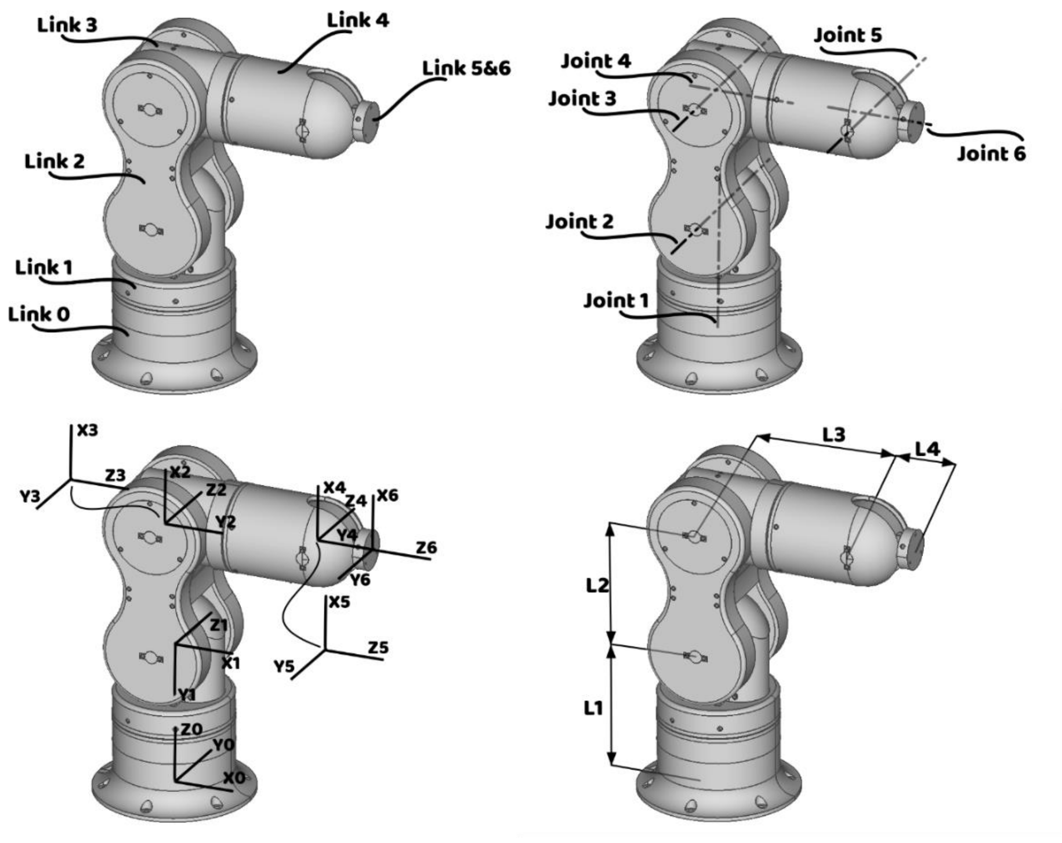


Fig. 11. Thor parameters representation [29]

Measurement of each part is shown in **Table 4**.

Table 4. Thor parameters values

Variable	**Value**
L1	202 mm
L2	160 mm
L3	195 mm
L4	67.5 mm
Gripper	175.5 mm

Knowing these data, the next step is to obtain its Denavit-Hartenberg parameters (see **Table 5**), which give us the mathematical model of the robot that can be used for subsequent calculations of direct and inverse kinematics. It has been added also a rotation of -90º to joint 3 ($\theta_3 = -90º$), so its initial position is completely vertical.

Table 5. Thor Denavit-Hartenberg table.

Joints	**θ**	**d**	**a**	**α**
Joint 1	0	L1	0	-90
Joint 2	-90	0	L2	0
Joint 3	-90	0	0	-90
Joint 4	0	L3	0	90
Joint 5	0	0	0	-90
Joint 6	0	L4 + Gripper	0	0

In addition to this, it will be also added to the model the minimum and maximum limits for the rotation of each of the joints, shown in **Table 6**.

Table 6. Thor‘s joints angles limits.

Joints	**Minimum(º)**	**Maximum(º)**
Joint 1	-180	180
Joint 2	-90	90
Joint 3	-90	90
Joint 4	-180	180
Joint 5	-90	90
Joint 6	-180	180

To verify the right working of the mathematical model and to have a virtual environment for simulating and testing the robots has been used RoboDK[30]. RoboDK is a software used for simulation and programming of industrial robots, and you can even create your own robots in it by placing the 3D model of each part and bringing it to life through the Denavit-Hartenberg parameter table. In this way, the Thor robotic arm model has been virtualized as shown in **Fig. 12**.

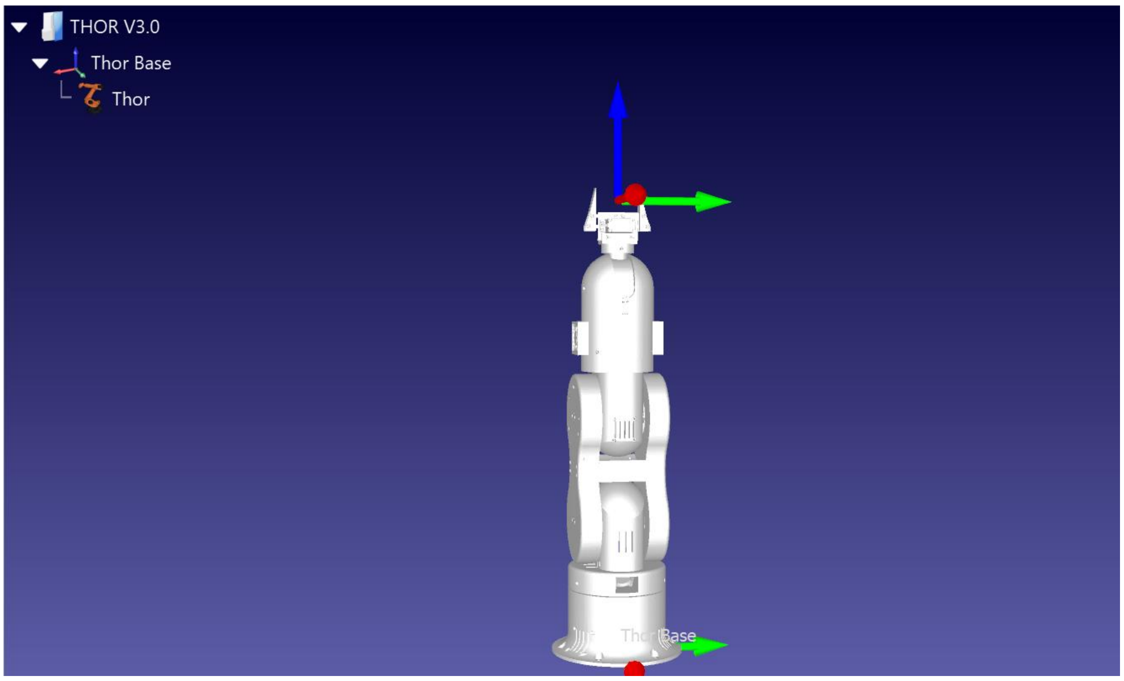


Fig. 12. Thor modeled in RoboDK

After verifying the correct working of the model, the arm was painted in two different ways, differentiating both arms that will be used later in the robotic station (see **Fig. 13**).

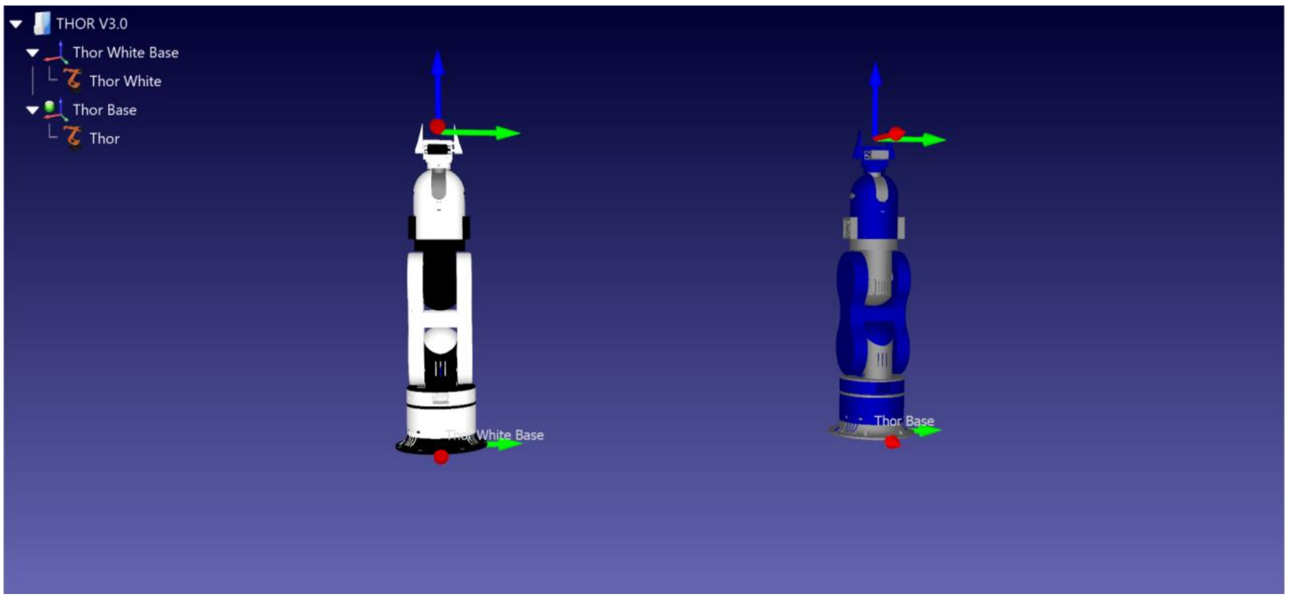


Fig. 13. Robotic arms's models painted

2.3. Robot components

2.3.1. Actuators

As actuators, there are 7 bipolar NEMA 17 stepper motors (17hs3401). Each joint uses one motor, except for joint two, which is formed by two of these motors. This is because when the arm is stretched to one side, the moment created by the weight of the arm cannot be moved using a single motor.

In addition to these, both motors in joint two and the motor in joint three will have a gearbox with a 5.18:1 gear ratio, providing the necessary torque for the motors to move the robot.

As an additional actuator, there is a 30kg strength servo motor, which is responsible for opening and closing the gripper. Due to its high-power consumption, a step-down to 6V 3A is supplying it.

2.3.2. Motor drivers

To control each stepper motor, it is necessary to use a motor driver. In this case, an A4988 module will be used to control each motor. These modules can function by performing full steps or microsteps of up to 1/16 steps. They are designed to work with bipolar stepper motors up to 35V and 2A.

2.3.3. Microcontrollers

As mentioned before, two microcontrollers are used to control each robotic arm. In the middle part of the arm an Arduino Nanocis use. It responsible for reading all the sensors data of the robotic arm and sending control signals to the gripper. This microcontroller was selected due to its small size and possessing the necessary GPIO pins and a UART channel to connect to the main controller.

On the other hand, at the bottom of the arm an Arduino Mega is used, which has four serial communication channels (UART), so it can communicate with both the robotic station and the controller in the middle of the arm, as well as having enough pins for connecting all components and sufficient power to control 7 motors simultaneously, which is why it is used in a wide variety of open source robotic arms.

2.3.4. Custom PCB

For the Arduino Mega, there are numerous shields capable of connecting up to 5 A4988 drivers for stepper motors, but no more. For this reason, has been assembled the custom PCB designed by Angel L.M. for the original THOR version. This PCB is an Arduino Mega shield that supports up to 8 motor drivers.

These boards consist of SMD and through-hole components, so for their assembly, SMD components were soldered first in an oven as shown in **Fig. 14**.

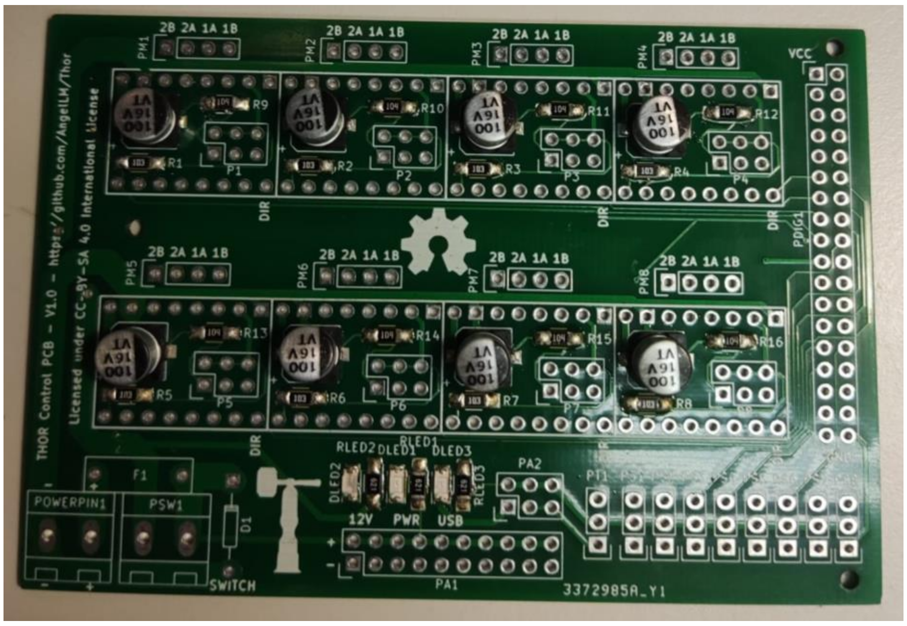


Fig. 14. Custom PCB with SMD soldered

Once they are soldered, a conventional soldering iron is used to finish soldering the remaining components (see **Fig. 15**).

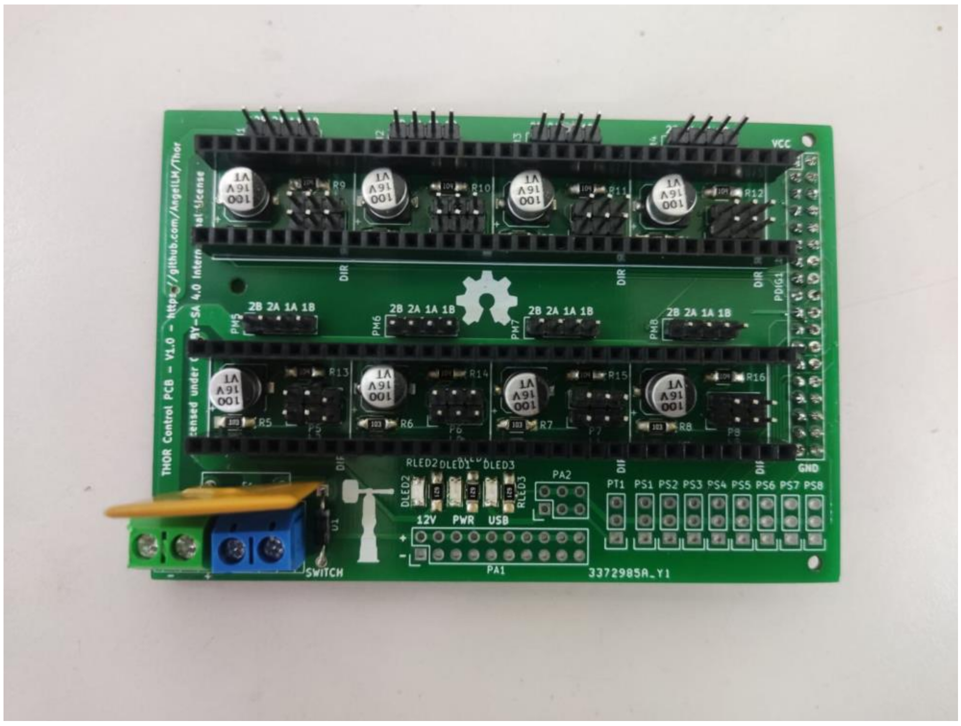


Fig. 15. Assembled custom PCB

2.3.5. Encoders

For the encoders, it has been used the same as the used by Danny VdH in the chosen robot model. Thus, multi-turn potentiometers will be use in joints 1 and 4, as these joints can rotate completely, and these encoders have a measurement range of 10 turns (3600º).

On the other hand, for joints 2, 3, 5, and 6, SMD rotary angle sensors will be used, which are potentiometers that can give values with a precision of 1º in a range of 330º, which is sufficient for the robot considering that these axes only move between 90º and -90º.

The exception to this would be axes 5 and 6, as the movements of these axes are combined. Thus, to rotate joint 6, both motors must rotate in the same direction, while to rotate joint 5, each motor must rotate in a different direction.

It results in the range of rotation shown in equation 2.3.

$$\theta_{min} = -90 - 180 = -270º \tag{2.1}$$

$$\theta_{max} = 90 + 180 = 270º \tag{2.2}$$

$$Range = 2 * 270 = 540º \tag{2.3}$$

But the feedback using these SMD encoders can only cover a range of 330º. Even so, due to design issues in the joint and lack of space, these limited encoders will be used. To solve the problem, the feedback function will be disabled when the robot goes to a configuration that exceeds this combined angle, such as when joint 5 rotates to its maximum value (±90º) and joint 6 to a value greater than ±75º.

2.4. Control system design

2.4.1. Selection of the central controller

Once have been decided which controller to use for each robotic arms, it is time to think about how both robots will be controlled in a coordinated way using a custom central GUI [31].

For this task, a Raspberry Pi 3 Model B has been used, because unlike a microcontroller such as Arduino, this is a computer, and it has enough power to run a graphical interface on a monitor and under an operating system like Linux.

In this case, the Raspberry Pi OS operating system has been installed on this Raspberry Pi, which is based on the Debian distribution of Linux and is optimized to run more efficiently on this type of mini-computer.

2.4.2. Selection of communication protocols

For fast and secure communication between the central controller and the main controller of each arm, and between the main and secondary controllers of an arm, it has been decided to use the UART serial communication protocol. Using this protocol you can create a connection between two devices, and both are capable of sending and receiving information.

In this way, the central controller will be able to send action commands to each arm, and they can send a message every time they finish to perform the action indicated by the central controller. Also, the main controller of the arm can order to the secondary controller to move the gripper or request the angle values of the joints, and the controller can respond by sending a message when finished or with the requested information. So there is a bidirectional communication between the controllers, with the security and efficiency offered by the UART serial communication at short distances.

On the other hand, the method of communication between the central controller and the screen where the graphical interface will be used must be chosen. The simplest method would be to connect a monitor using an HDMI cable, as the Raspberry includes an HDMI connector.

However, the idea of the robotic station includes a wireless screen, so you can control both robots even at a certain distance. Because of it, it has been decided to use a commercial tablet, as it includes a screen, battery, and internet connection for a lower price than it would cost to build something similar from scratch. The chosen tablet has been a Lenovo Tab M10 Plus (3rd gen).

And as a communication method, it has been decided to use TeamViewer, which is a solution created for remote control of computer devices and can be installed on both the Raspberry Pi and the tablet.

These types of programs usually have a significant delay[32], so a hotspot has been created on the Raspberry, so that the tablet connects to this network, using TeamViewer through the local network and eliminating the delay.

2.4.3. Control diagram

The control diagram of the robotic station is shown in **Fig. 16**.

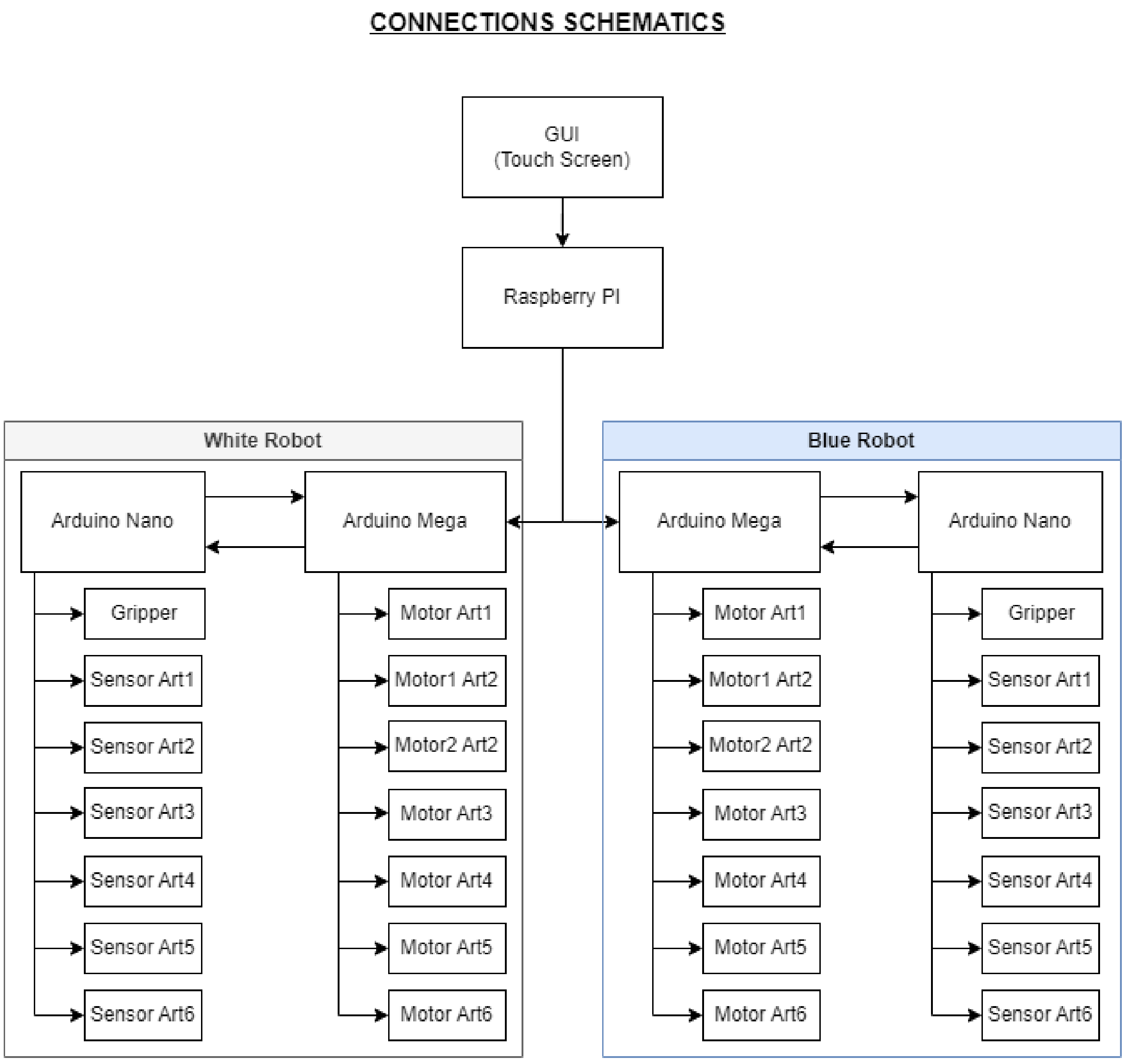


Fig. 16. Robotic station‘s control schematic

2.5. Power system design

2.5.1. Calculation of required power

Once the structure of the robotic station has been decided, the next step is to select a power supply with enough power for the system. To do this the maximum power that each part of the system can consume must be calculated.

Voltage and maximum current of each device is shown in **Table 7**.

Table 7. Voltage and maximum current comsumed by the components

Components	**Voltage (V)**	**Current (A)**
Raspberry PI	5	2.5
Tablet	5	3
Arduino Mega	5	0.5
Arduino Nano	5	0.5
Servomotor	4.8-7.2	3A
Stepper-motor 17S3401	12	1.3A

As it can be noticed, it's needed to supply different voltage levels, from 12V to 5V. Therefore, a 12V power supply will be use, as it is the maximum voltage level required in the station; and the rest of the voltage levels will be obtained through the use of step-down circuits.

The first step is to calculate the maximum power consumed by the stepper motors. Each arm has 7 17S4401 motors, which gives the consumption shown in equation 2.6.

$$P_{motors} = V_{motor} \cdot I_{motor} \cdot n_{motors} \tag{2.4}$$

$$P_{motors} = 12 \cdot 1.3 \cdot 7 \tag{2.5}$$

$$P_{motors} = 109.2\ W \tag{2.6}$$

For the rest of the components in the system, it is needed to use step-down circuits to obtain the different voltage levels, considering that these circuits have an efficiency of around 90%.

In the **Table 8** is written the power required for each component. The Arduinos Mega and encoders are not included, as the Arduinos Mega are powered from the USB ports of the Raspberry Pi 3, as each USB port can provide up to 600mA. The encoder consumption is accounted for within the consumption of the Arduino Nano.

Table 8. Components's consumption

Components	**Voltage(V)**	**Current(A)**	**Power(W)**	**Power/0.9(W)**
Raspberry PI	5	2.5	12.5	13.89
Tablet	5	3	15	16.67
Servomotor	6	3	18	20.00
Arduino Nano	5	0.5	2.5	2.78

In the robotic station there are two robotic arms, so the calculated consumption of the motors, servo motor, and Arduino Nano will need to be doubled. The total consumption of each robotic arm is shown in equation 2.9.

$$P_{robot} = P_{motors} + P_{servo} + P_{nano} \tag{2.7}$$

$$P_{robot} = 109.2 + 20 + 2.78 \tag{2.8}$$

$$P_{robot} = 131{,}98\ W \tag{2.9}$$

Therefore, the maximum consumption of the system is calculated in equation 2.12.

$$P = P_{RP3} + P_{Tablet} + 2 \cdot P_{robot} \tag{2.10}$$

$$P = 13.89 + 16.67 + 2 \cdot 131{,}98 \tag{2.11}$$

$$P = 294.52W \tag{2.12}$$

Now, applying a safety factor of 20% the new power consumption is shown in equation 2.14.

$$P_{total} = P \cdot 1.2 = 294.52 \cdot 1.2 \tag{2.13}$$

$$P_{total} = 353.42\ W \tag{2.14}$$

It implies that the 12V source must be able to supply at least 29.45 A as shown in equation 2.17.

$$I_{min} = \frac{P_{total}}{V} \tag{2.15}$$

$$I_{min} = \frac{353.42}{12} \tag{2.16}$$

$$I_{min} = 29.45A \tag{2.17}$$

Finally, it must be calculate the diameter of the cable necessary to efficiently supply current to each robot. To do this is used the equation 2.18.

$$A = \frac{I \cdot \rho \cdot L}{fk \cdot V} \tag{2.18}$$

Where ρ is the resistivity of the conductor material, in this case copper (0.0175 **Ω**·mm^2/m), fk is the loss factor that is assumed as 1% (fk=0.01), and L is the length of the cable, in this case 0.5m. Additionally, the maximum consumption of each arm is about 132W, which is equivalent to a voltage (V) of 12V and a current (I) of 11A, which, multiplied by a safety factor of 1.2, becomes 13.2A. Therefore, the minimum cable section will be the shown in equation 2.20.

$$A_{min} = \frac{13.2 \cdot 0.0172 \cdot 0.5}{0.01 \cdot 12} \tag{2.19}$$

$$A_{min} = 0.946\ mm^2 \tag{2.20}$$

2.5.2. Components selection

After the calculation of the power needed by the system, the next step is to choose the components. In this section, the selection of the power supply will be discussed, cables, and step-down circuits.

The first thing is to select a power supply that provides at least 353W. In this case, a 360W power supply that was already available has been selected, which has three outputs at 12V and 10A each, giving a total of 30A.

To avoid power failures, it has been decided to use all the outputs of this power supply, also making use of a multimeter module that is connected to the power supply and provides us with a real-time reading of the voltage, current, and power always consumed. Additionally, to achieve the lower voltage levels it will be use an adjustable step-down regulator module.

As mentioned earlier, the step-down circuit used has an energy efficiency of around 90% and can supply up to 3A, so several modules are needed. On the one hand, two will be needed in the central box with a 5V output, to power the Raspberry Pi and to charge the Tablet. And on the other hand, two will be needed in each robot, one exclusively for the servo motor, due to its high consumption, which will decrease the voltage to 6V, and another for the Arduino Nano and encoders, with a 5V output. Therefore, a total of 6 step-down modules is needed.

By modularizing the step-down in this way, the current transmission between blocks, that is, from the central box to each of the arms, is carried out directly from the 12V power supply, using cables with a diameter of 1.5mm^2, fulfilling the minimum requirement calculated above.

2.5.3. Power connections diagram

The power connections of the robotic station are shown in **Fig. 17**.

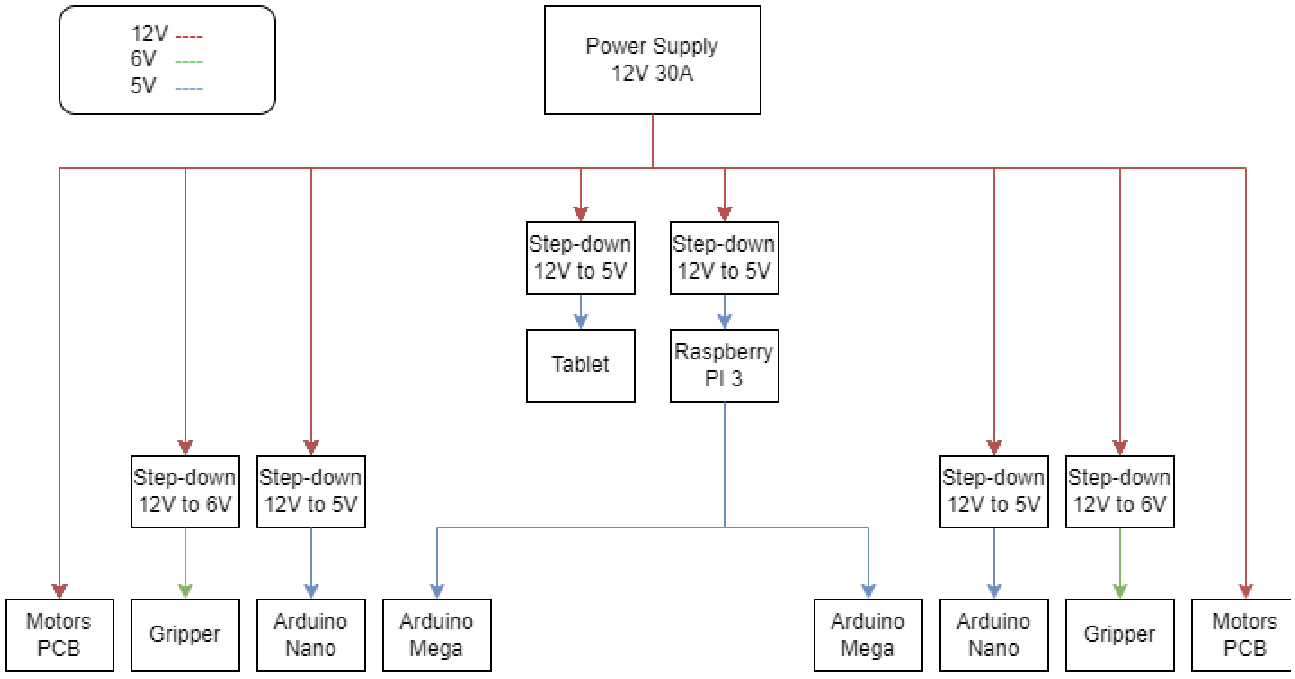


Fig. 17. Robotic station‘s power schematic

2.6. Connections diagram of the robotic station

The connections diagram of the robotic station is shown in **Fig. 18**.

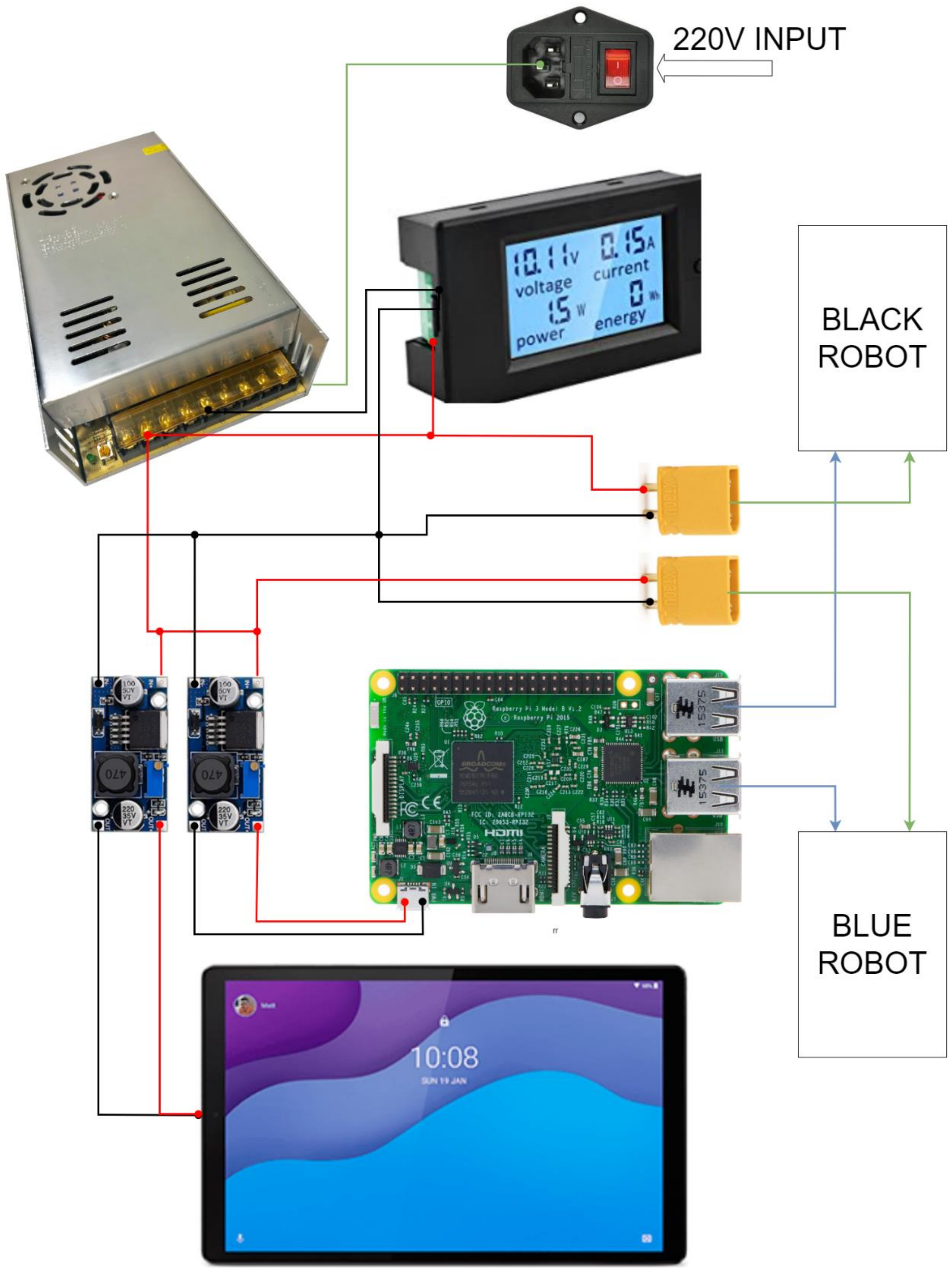


Fig. 18. Robotic Station‘s connections diagram

The intern connections diagram of each robot is shown in **Fig. 19**.

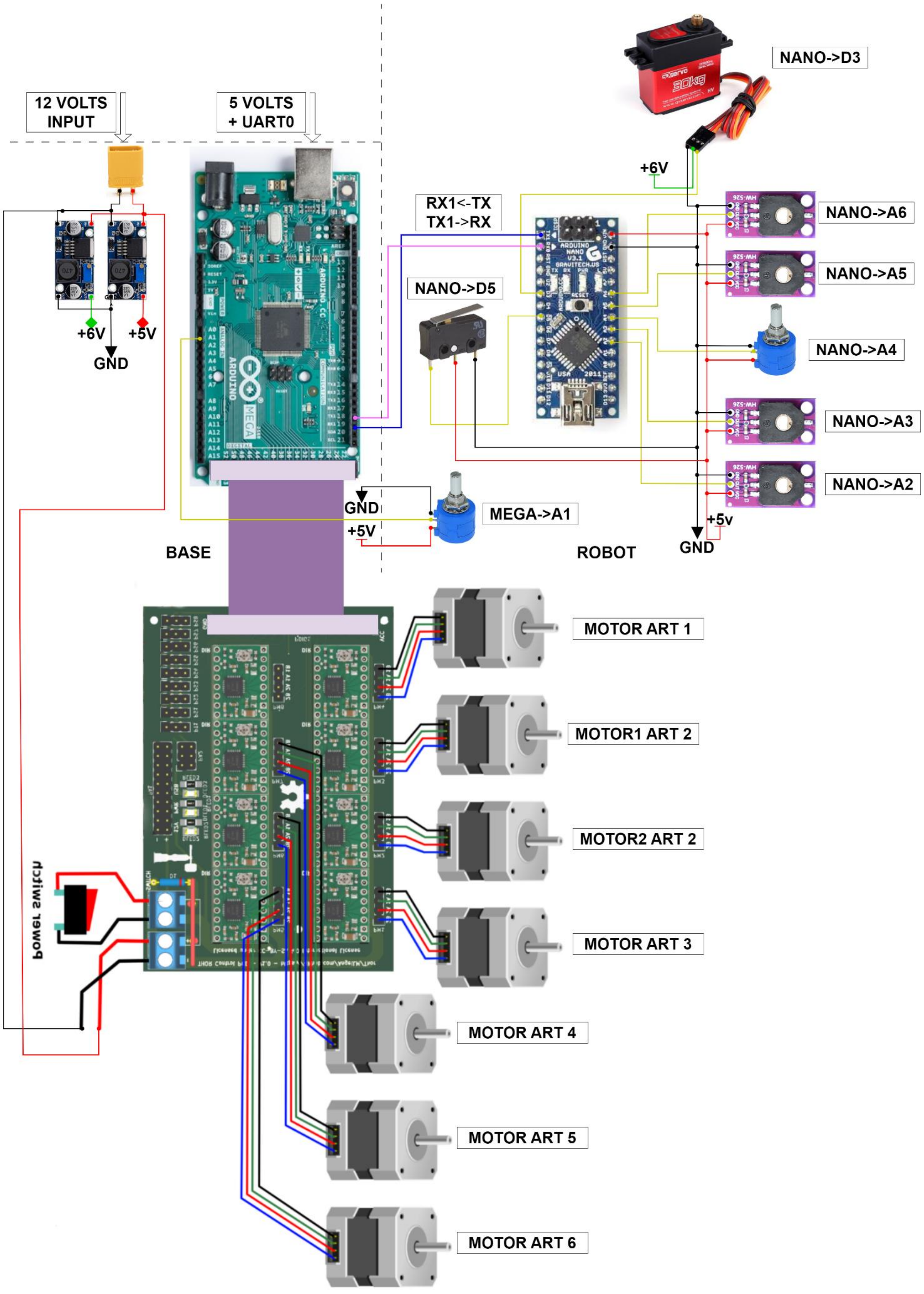


Fig. 19. Robot‘s connections diagram

2.7. Robotic station structure

Therefore, the robotic station will be divided and encapsulated into three blocks: a central box and two robots. On the one hand, the central box will contain the power supply, the central controller, and a support to place the tablet. On the other hand, each robot will have a box on its base where its microcontroller, two step-down modules, and motor drivers are located. In this way, each robot has only two inputs, the power input at 12V and a USB input for communication and powering of the Arduino Mega.

Thus, it is obtained a highly modular design, being able to take a robot from the station and make it work by supplying only 12V and sending orders through the USB port from any computer. In addition to this, two supports for the robotic arms have been added, which can be attached and removed, so the arms can be held on these supports when they are not in use.

With all the components clear, the only thing left to decide is how the arms will be placed in the robotic station for having enough workspace and at the same time, cooperation capacity. To make this decision, RoboDK has been used, the simulation software mentioned above. It was used by placing both robots on the same table to check the range of movements they have and to search the optimal distance between them, with a common and an individual space (see **Fig. 20**).

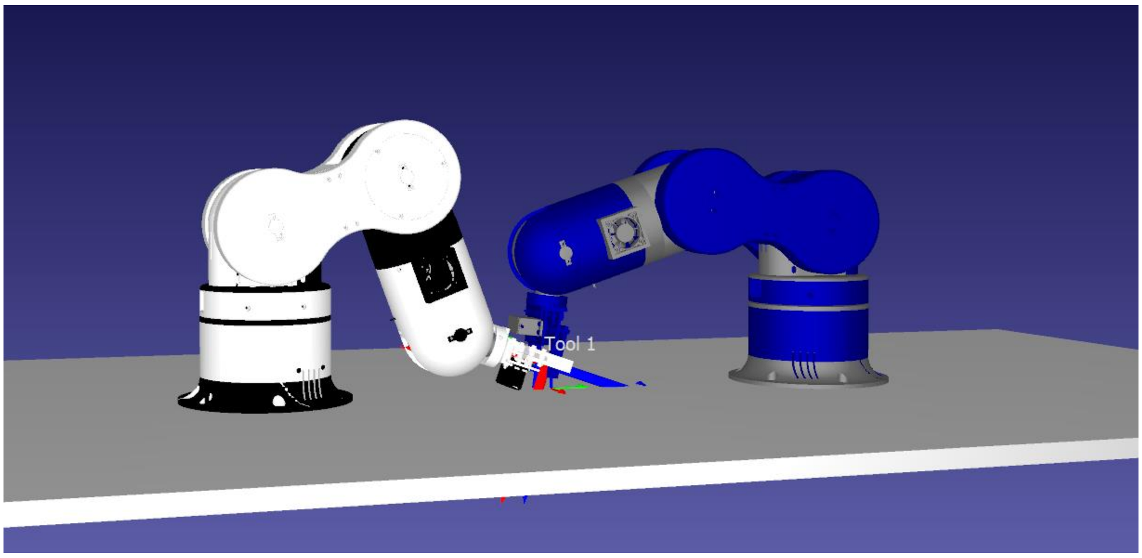


Fig. 20. Robotic station simulation

After some tests, it was decided that the best distance would be 700m between the center of the robots, since each robot has a range of approximately 420mm, and in this way, there is a central area in which both robots can manipulate objects with different configurations. If the robots were further they would not be able to pick up objects by placing the gripper perpendicular to the ground in the common workspace. Once the distance between robots was decided, all components were distributed and ordered so that the table on which will be assembled the robotic station don't become excessively large, obtaining as total dimensions of 950x750mm.

2.8. Robotic Station flowcharts

The flowchart of the robotic station is shown in **Fig. 21**.

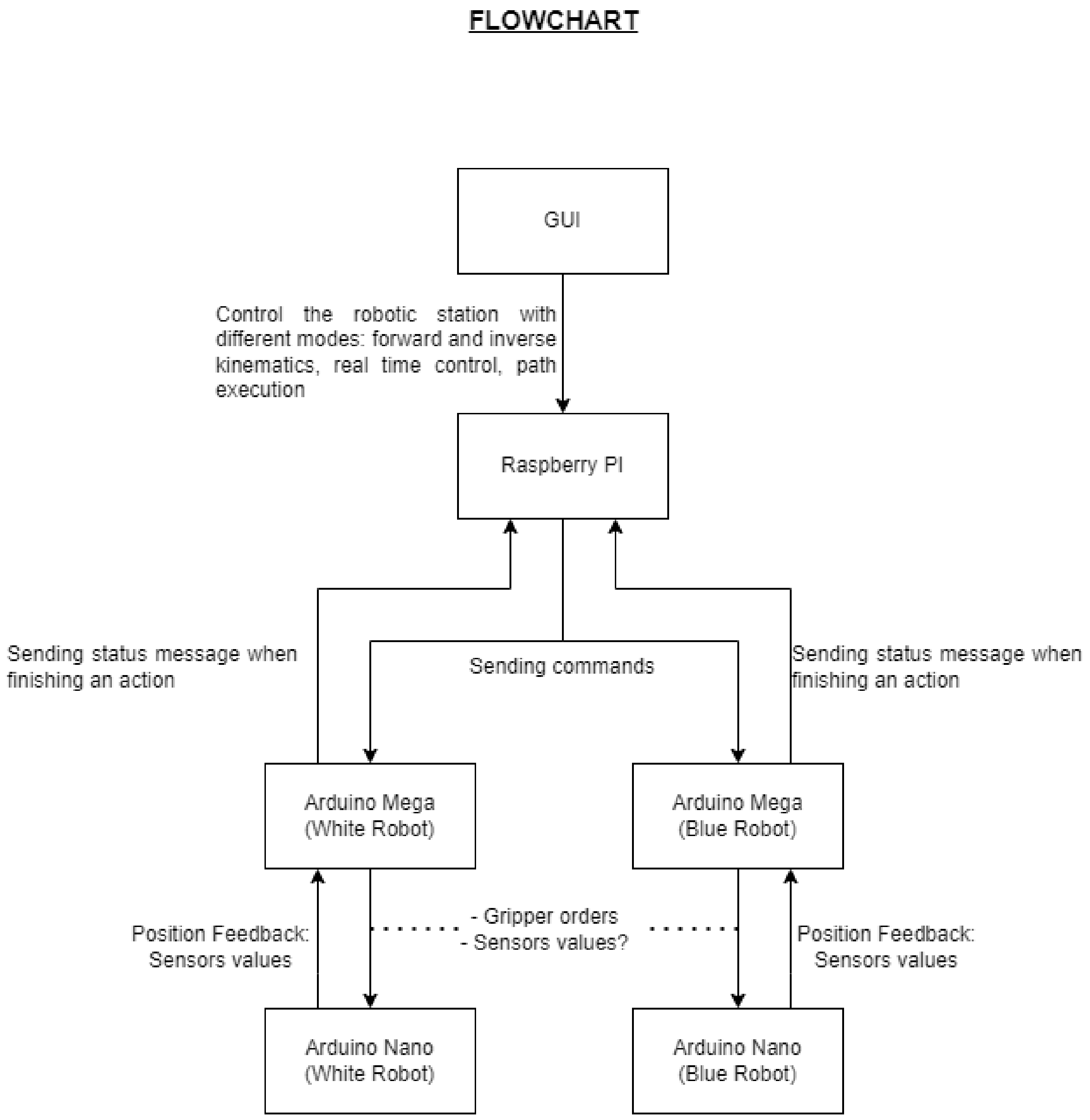

FLOWCHART
GUI
Control the robotic station with different modes: forward and inverse kinematics, real time control, path execution
Raspberry PI
Sending status message when finishing an action
Sending commands
Sending status message when finishing an action
Arduino Mega (White Robot)
Arduino Mega (Blue Robot)
Position Feedback: Sensors values
- Gripper orders
- Sensors values?
Position Feedback: Sensors values
Arduino Nano (White Robot)
Arduino Nano (Blue Robot)

Fig. 21. Robotic Station flowchart

The robotic station (**Fig. 21**) is formed by 3 different types of controllers, the central controller and two controllers per robot, the main one and the secondary, in charge of the data of the encoders and control the gripper. In the next flowcharts it will be shown the flowchart of each controller.

2.8.1. Central controller

The flowchart of the central controller is shown in **Fig. 22**.

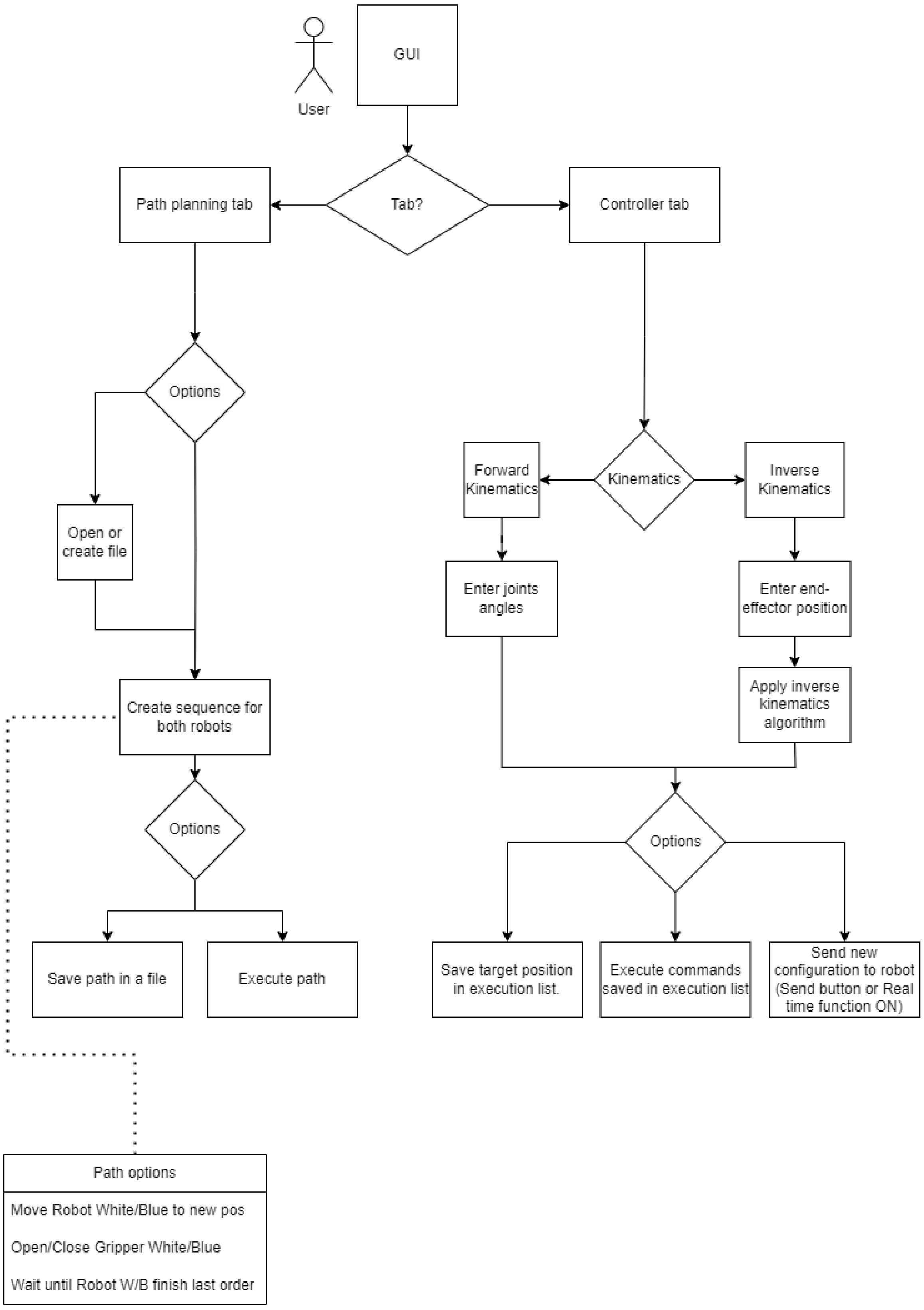

User
GUI
Tab?
Path planning tab
Controller tab
Options
Open or create file
Create sequence for both robots
Options
Save path in a file
Execute path
Kinematics
Forward Kinematics
Inverse Kinematics
Enter joints angles
Enter end-effector position
Apply inverse kinematics algorithm
Options
Save target position in execution list.
Execute commands saved in execution list
Send new configuration to robot (Send button or Real time function ON)
Path options
Move Robot White/Blue to new pos
Open/Close Gripper White/Blue
Wait until Robot W/B finish last order

Fig. 22. Central controller flowchart

2.8.2. Robot controller

The flowchart of the robot controller, the main controller of the arm, is shown in **Fig. 23**.

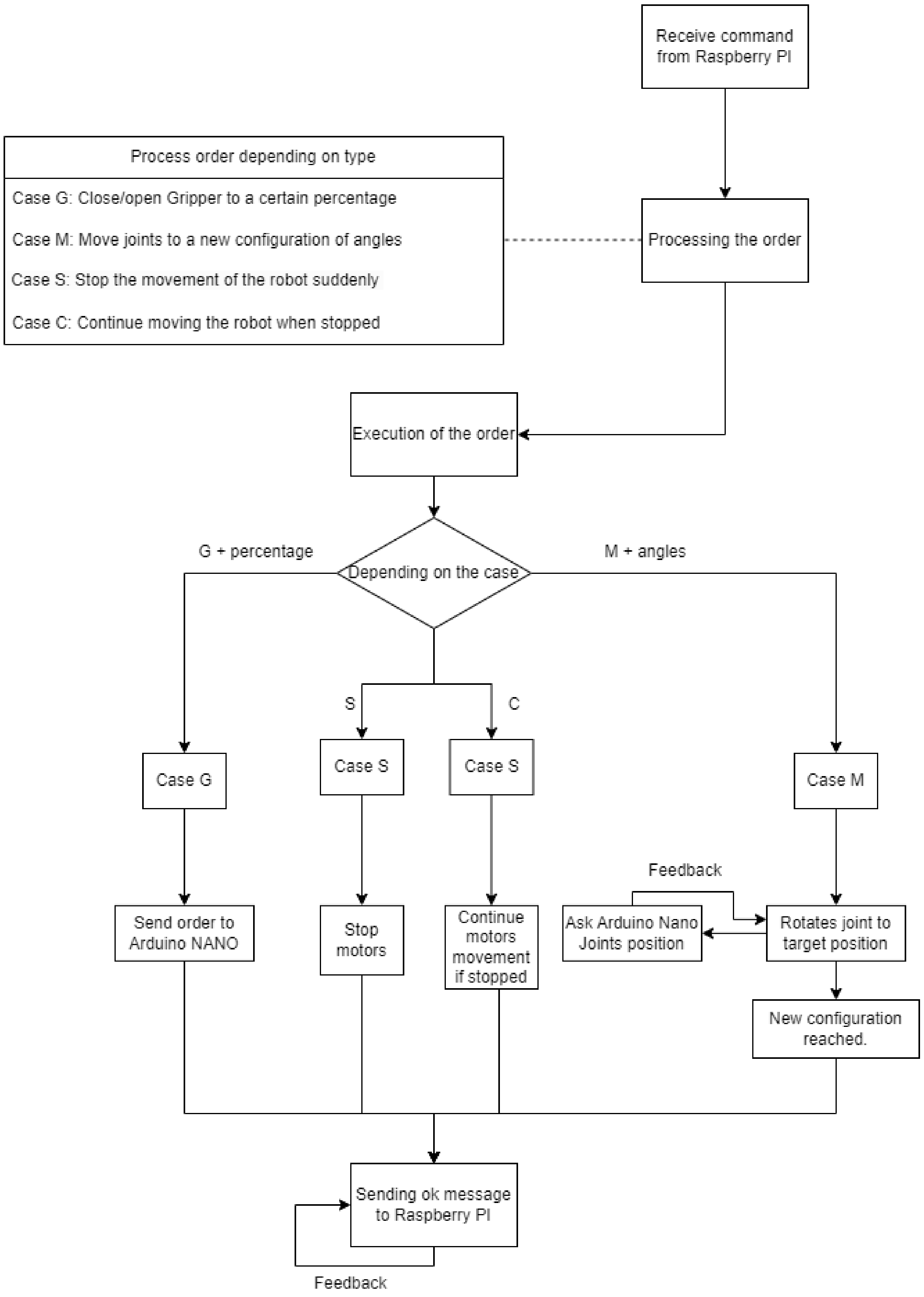
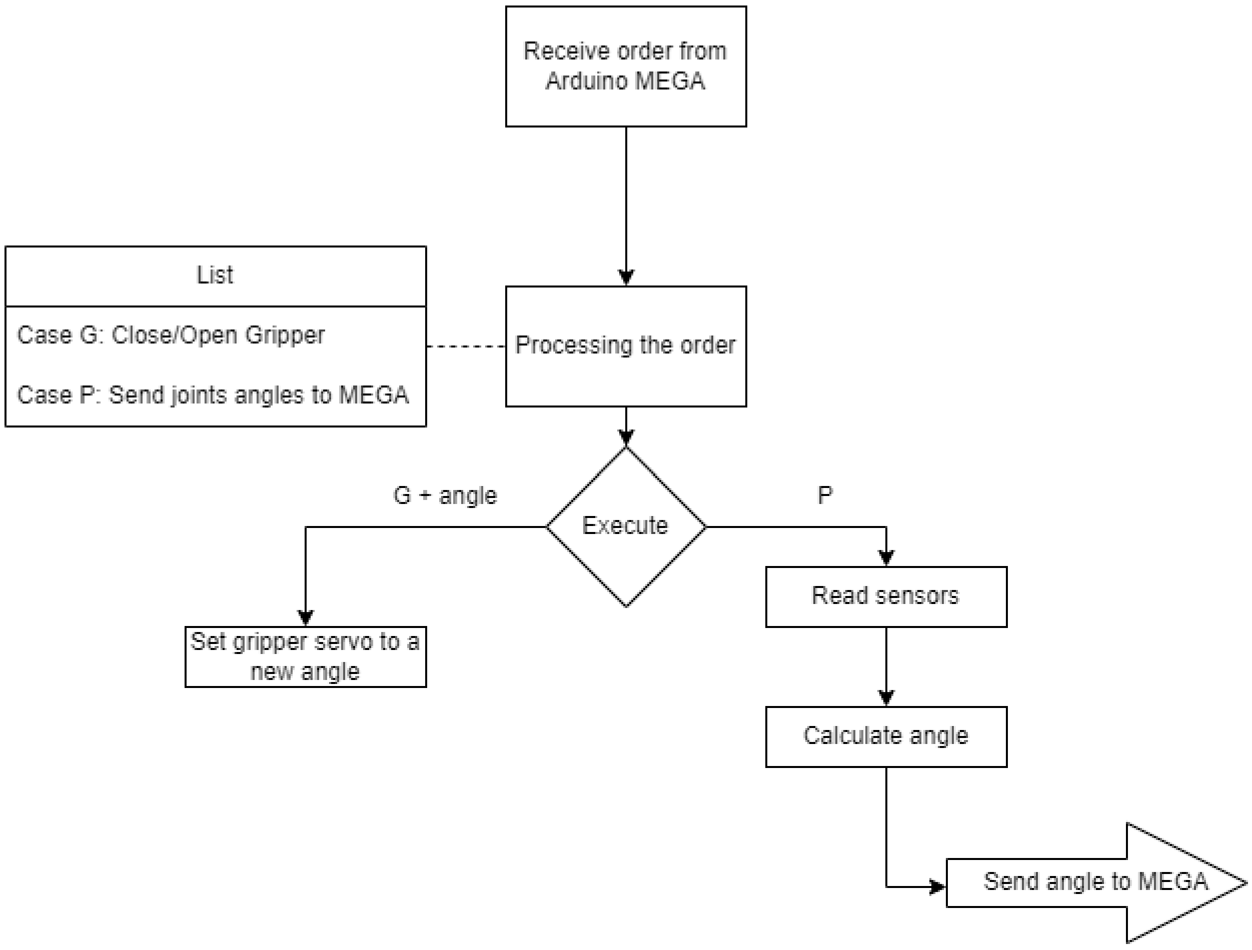

Receive command from Raspberry PI
Process order depending on type
Case G: Close/open Gripper to a certain percentage
Case M: Move joints to a new configuration of angles
Case S: Stop the movement of the robot suddenly
Case C: Continue moving the robot when stopped
Processing the order
Execution of the order
Depending on the case
G + percentage
M + angles
S
C
Case G
Case S
Case S
Case M
Feedback
Send order to Arduino NANO
Stop motors
Continue motors movement if stopped
Ask Arduino Nano Joints position
Rotates joint to target position
New configuration reached.
Sending ok message to Raspberry PI
Feedback

Fig. 23. Robot controller flowchart

2.8.3. Encoder controller

The flowchart of the encoders controller, the secondary controller of the arm, is shown in **Fig. 24**.



Receive order from Arduino MEGA
List
Case G: Close/Open Gripper
Case P: Send joints angles to MEGA
Processing the order
G + angle
Execute
P
Read sensors
Set gripper servo to a new angle
Calculate angle
Send angle to MEGA

Fig. 24. Encoder controller flowchart

3. Experimental approach

3.1. Assembly of the central box

The first step was testing the power supply with the incorporated multimeter (see **Fig. 25**).

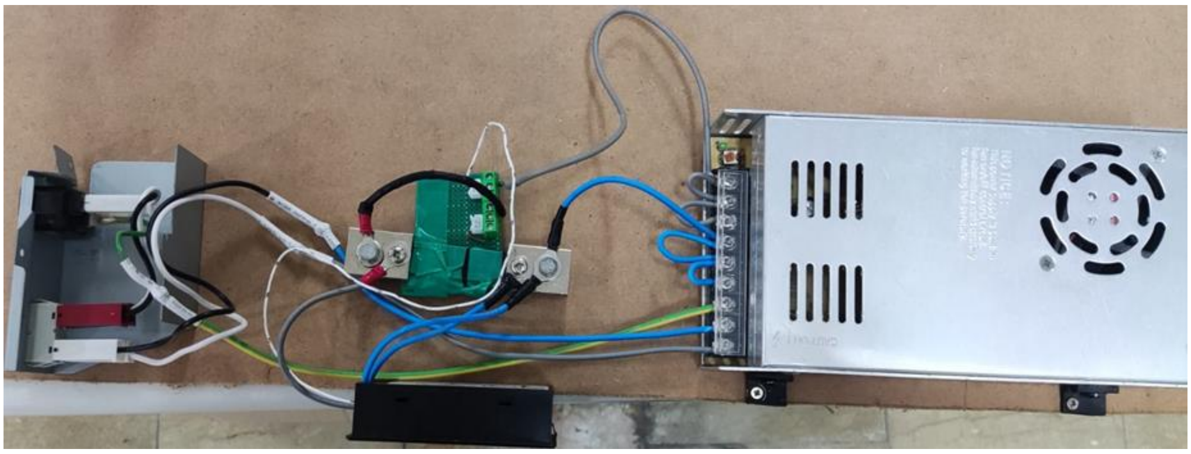


Fig. 25. Power supply test

Once the multimeter is working it has been added the Raspberry Pi with two fans for air flow, an emergency button, two internal step-down modules, for powering the raspberry pi and charging the tablet, and two usb from Raspberry Pi and two 12V outputs for the robotic arms, as shown before in the connections diagram (**Fig. 19**). Then it is assemblied in the central boxas shown in **Fig. 26**.

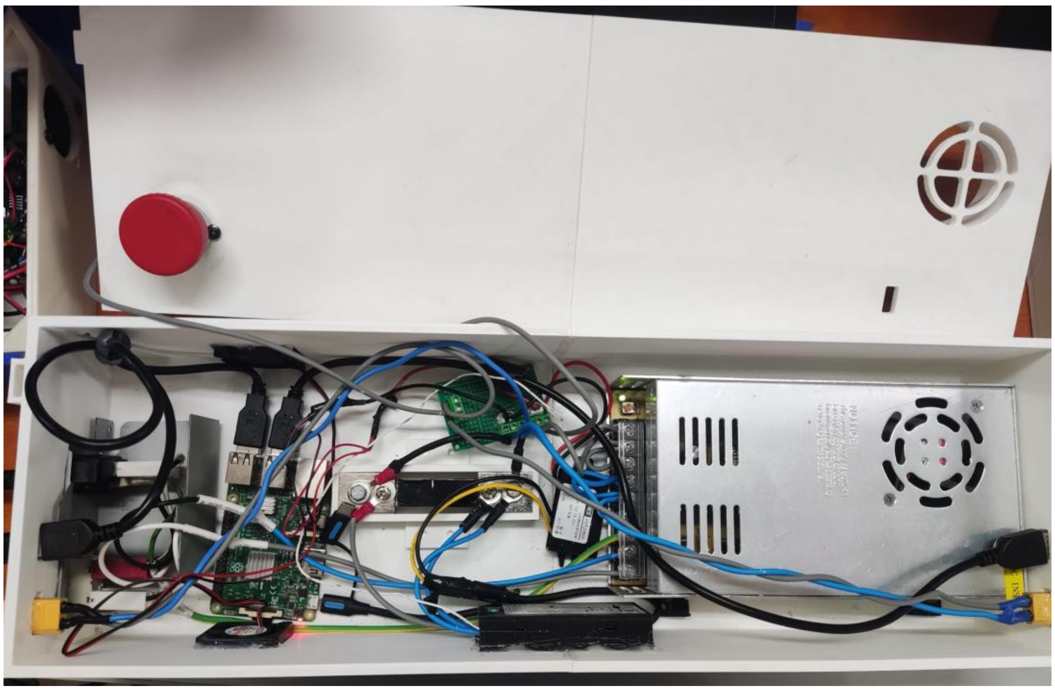


Fig. 26. Central box assembly

3.2. Assembly of robots

The first step is to assembly the joints five and six, as shown in **Fig. 27**.

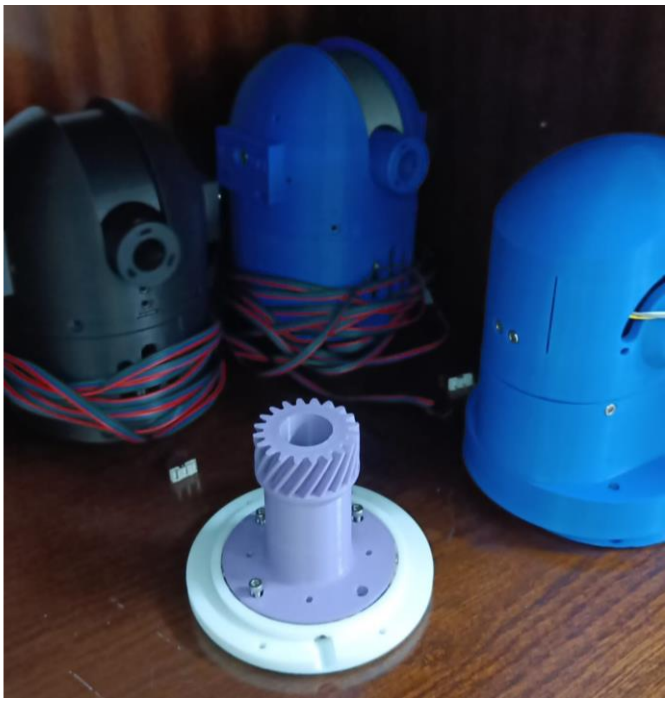


Fig. 27. Fifth and sixth joint of each robot assemblied

Once both joints were tested moving correctly from the Arduino the joint 4 was added (see **Fig. 28**).

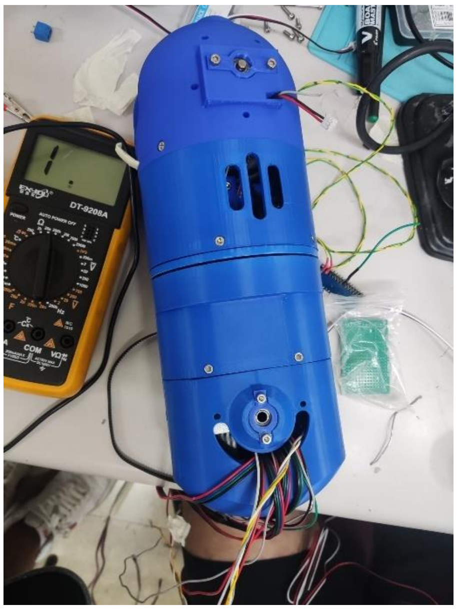


Fig. 28. Robot assemblied from the forth joint to the sixth joint

With them working it was prepared the base with the joint1 as shown in **Fig. 29**.

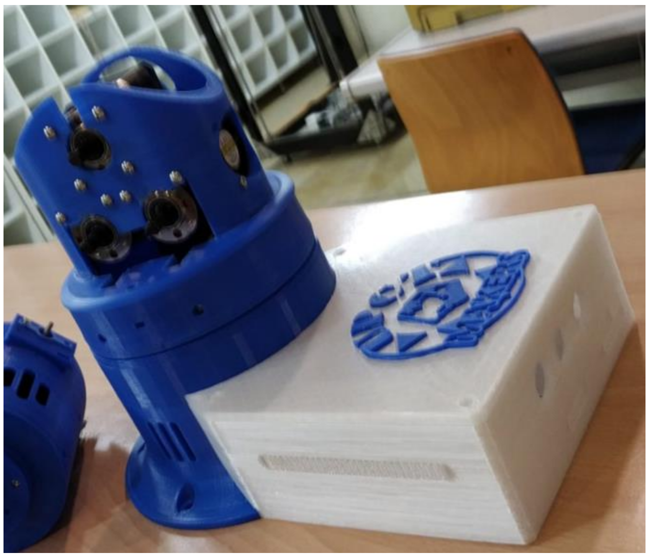


Fig. 29. Base and first joint assemblied

The next step consists in adding the second and third joints. In this step the encoders controller has to be added, connecting every the gripper, encoders, power supply and serial communication wires as shown in the diagram of **Fig. 19**. The result is shown in **Fig. 30**.

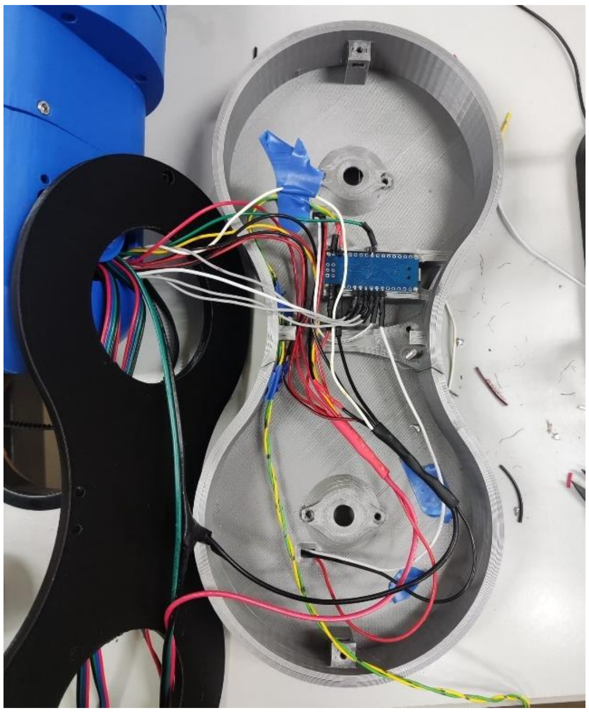


Fig. 30. Encoders controller connections

After realizing the connections the piece can be added to the forth joint as shown in **Fig. 31**.

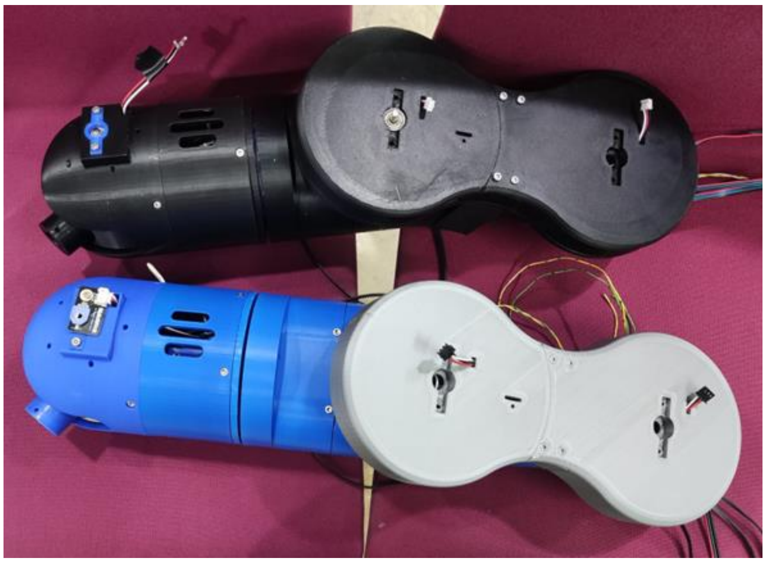


Fig. 31. Robots from third to sixth joint

Finally the higher parts of the robot shown in the **Fig. 31** are attached to the lower part shown in **Fig. 29**, and the gripper can be added, finishing the assembly of the robotic arm (see **Fig. 32**).

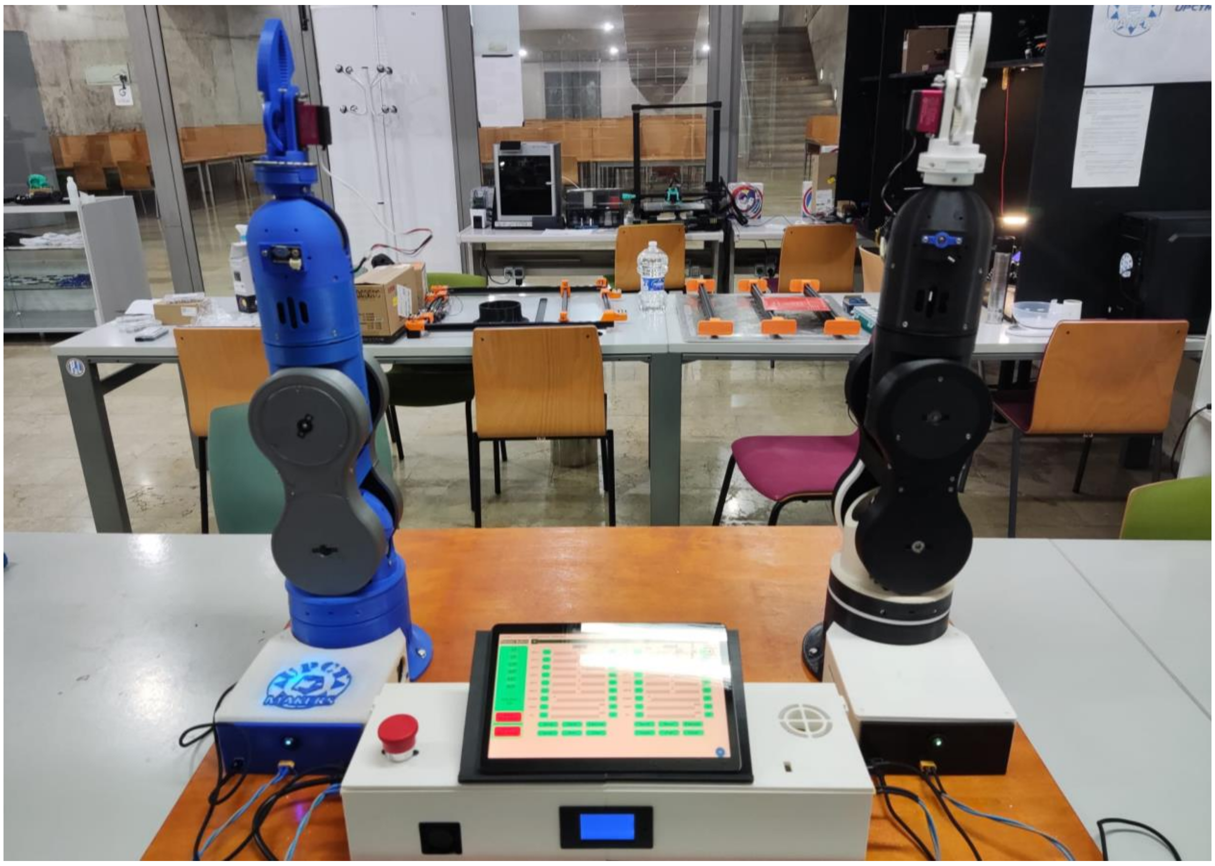


Fig. 32. Robots assemblied

3.3. Assembly of robots boxes

With the arms assemblied is time for finishing its electronnics boxes. First of all the A4988 drivers must be prepared, seting the maximum current that the driver will provide to the motors. In the case of the project motors its maximum current is 1.5A, so the limito f the drivers will be setted to 1.2A.

As shown in the page 9 of its datasheet[33] for setting the driver to this maximum current value a test must be done, adjusting the driver potentiometer to measure a determined voltage value, calculated with the equation 3.1.

$$V_{REF} = I_{TripMax} \cdot 8 \cdot R_S \tag{3.1}$$

In the case of our module $R_S = 0.1\ \Omega$, V_{REF} is 0.96V as shown in equation 3.3.

$$V_{REF} = 1.2 \cdot 8 \cdot 0.1 \tag{3.2}$$

$$V_{REF} = 0.96\ V \tag{3.3}$$

With the reference voltage calculated the next step is to connect the VDD and GND pins of the drivers to the 5V and pins of the arduino in that order. Finally the multimeter negative pin is connected also to GND and the positive pini s connected to a screwdriver, which will be use for configurate the potentiometer value (see **Fig. 33**), achieving the desired reference voltage in the multimeter.

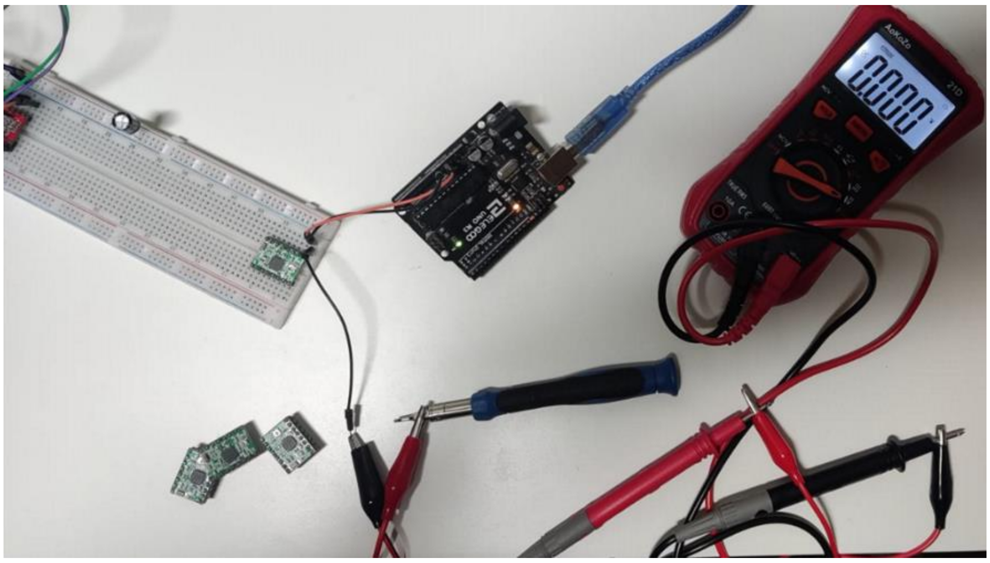


Fig. 33. Drivers A4988 maximum current test

After the tests the robots boxes have been assemblied as shown in Fig. 34, following its schematic (see **Fig. 19**), and everything works as expected.

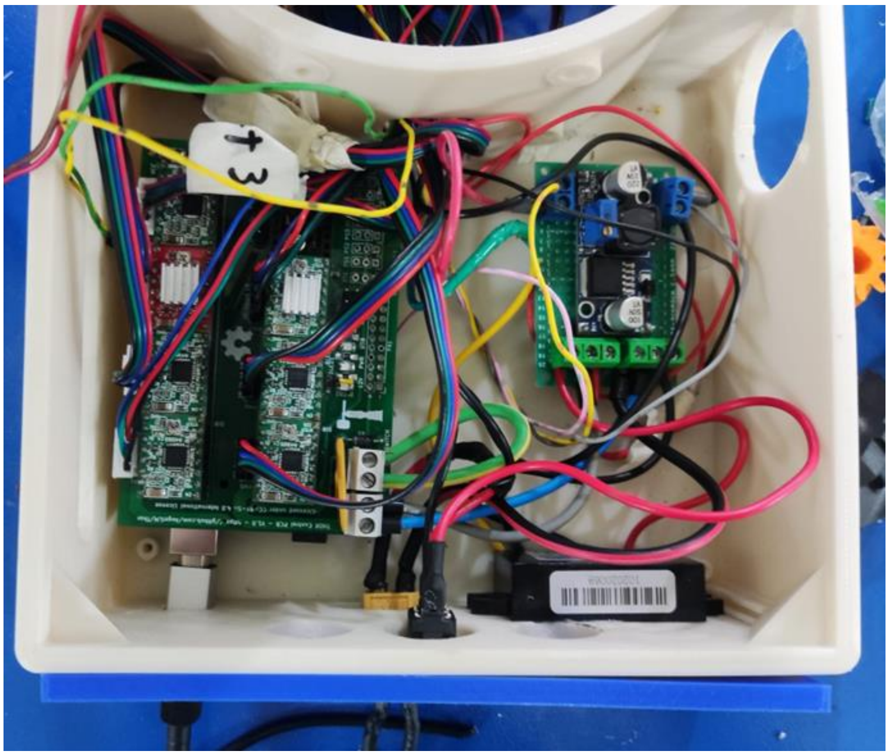


Fig. 34. Top view of the Blue Robot box

As shown in Fig. 35 each robot has two connections, an usb for the serial communication between the central controller and the robot and a power input of 12V, which can be openned using a switch.

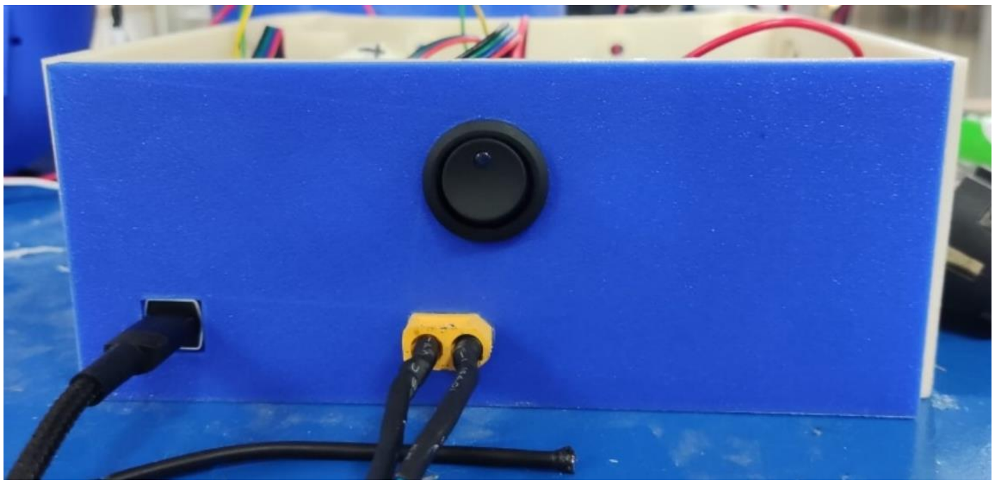
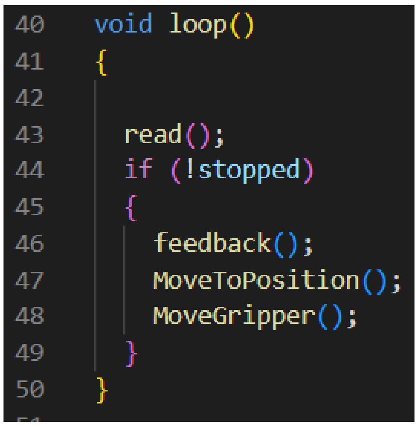


Fig. 35. Front view of the Blue Robot box

3.4. Programming of the robot controller

3.4.1. Main loop

The main loop of the robot controller follows a simple structure. It realizes 4 different actions shown in **Fig. 36**.



```
void loop()
{

  read();
  if (!stopped)
  {
    feedback();
    MoveToPosition();
    MoveGripper();
  }
}
```

Fig. 36. Robot controller main loop

First of all it reads if there have been new commands from the central controller. After if the controller didn‘t send a stop order the robot realizes a feedback for updating the joints position. Lastly it moves the axis and the gripper if needed.

Besides inside of the MoveToPosition function the controller is all the time checking if it has received new orders from the central controller (so it‘s possible to stop the robot without delay, improving the security of the system) and also it makes a feedback before of telling to the central controller that the position has been reached, moving the axis til approach the setpoint.

All these functions will be explained in detail in the next sections.

3.4.2. Serial communication with central controller

This communication is based in the next concept: Central controller sends commands, that can be movement commands or orders to stop or to continue the movement. And Arduino Mega answers to the central controller with the message, “Ok“ each time that the action has been performed.

In this way our controller will update the robot position each time it finishes his movements, and thanks to the feedback it is accurate with the reality.

This action is performed by the function read(). This function checks if there is a new line to read in the serial buffer, and if it is then the string is divided by the white spaces in an array of tokens.

The first of these tokens will tell us the type of action that the robot has to perform (see **Table 9**).

Table 9. Possible command‘s types by the central controller

token[0]	**Action**
‚M‘	Move joints to a new configuration of angles
‚G‘	Open/Close gripper in a specified percentage
‚S‘	Stop the movement of the robot suddenly
‚C‘	Continue moving the robot
‚P‘	Request for sending the current position
‚D‘	Deactivate the motors
'A'	Activate the motors

If the token[0] is ‚M‘ then the next 6 tokens are the new angles for the 6 joints, starting from the base, and the next tokens could be ‚S‘ or ‚A‘, what means that the next token to each of these letters will be the maximum speed and the acceleration for moving the joints. The structure of the command is shown in **Table 10**.

Table 10. Movement command example

pos[0]	**pos[1]**	**pos[2]**	**pos[3]**	**pos[4]**	**pos[5]**	**pos[6]**	**pos[7]**	**pos[8]**	**pos[9]**	**pos[10]**
M	0	0	0	0	0	0	S	2000	A	1000

In this example the robotic arm will move to the homing position {0,0,0,0,0,0} with a maximum speed of 2000 steps/s and an acceleration of 1000 steps/s^2.

If token[0] is ‚G‘ the next token will be the percentage of oppening of the gripper (see **Table 11**).

Table 11. Gripper command example

pos[0]	**pos[1]**
G	60

In this case the gripper will open itself to the 60% of its maximum.

If the token[0] is ‚S‘ then the robot will stop its movement and will remain waiting for the next order.

If the token[0] is ‚C‘ then the robot will continue its movement. If between the stop and continue the robot receives a new movement order it will start moving now to the new setpoint instead of the last one.

If the token[0] is ‚P‘ then the robot will execute the feedback() function and will send its current position to the central controller.

If the token[0] is ‚D‘ then the robot will deactivate the motors, doing it possible for you to move them freely using the hands.

If the token[0] is ‚A‘ the motors will be activated again.

Finally the robot will send an ‚Ok‘ message each time these actions are completed, except of the move action, that won‘t send this message til the feedback() match the setpoint.

3.4.3. Robot movement

To move the robot joints has been used the accelstepper library mentioned before. Using it all the motors have been put together in the same group using the multistepper extension of this library. In this way when i set the new positions of several motors they will move all together, mapping their velocities for finishing all the movements at the same time.

For achieve this the motor that has to realize the longest movement moves with the maximum speed chosen in the command, and the others motors map their speed to finish at the same time.

Once the motors have been added the working of the library is easy. You can set maximum speed, acceleration and the desired position of each motor. After this executing the function run() all the motors will move a step if needed, depending on its speed and position, while this function will be summing up the steps to the current position of the motor. Therefore a loop can be realised, executing the function run() while checking between each step if a new message have been received from the central controller.

When the run() function returns false it means that the setpoint has been reached. Then the feedback() function is executed for checking if it is true or our robot has lost some steps due to a collision or a big effort that resulted in a wrong position.

Also in the case of this robot the joints 5 and 6 don‘t have their own motors, so an extra calculation must be done for reaching the positions, and the movement must be done in two steps. First the joint 6 will rotate if needed, and after it the other 5 joints will rotate simultaneously.

When both motors rotates in the same direction the joint 6 rotates, and when both motors rotate in opposite directions the joint 5 rotates. To move this joints first it must be known in which position are they, using equations 3.4 and 3.5.

$$Angle_{joint6} = (angle_{motor5} + Angle_{motor6})/2 \tag{3.4}$$

$$Angle_{joint5} = angle_{motor5} - Angle_{joint6} \tag{3.5}$$

Once the current angle of the joint 6 has been calculated is done the difference with the new setpoint (see equation 3.6).

$$x = Setpoint_{joint6} - Angle_{joint6} \tag{3.6}$$

And the calculation of the new motors setpoints are calculated with equations 3.7 and 3.8.

$$angle_{motor5} = angle_{motor5} + x \tag{3.7}$$

$$angle_{motor6} = angle_{motor5} + x \tag{3.8}$$

Once this joint has been moved the calculations to move the joint 5 are performed with the rest of the joints using from equation 3.9 to equation 3.11.

$$x = Setpoint_{joint5} - Angle_{joint5} \tag{3.9}$$

$$angle_{motor5} = angle_{motor5} + x \tag{3.10}$$

$$angle_{motor6} = angle_{motor5} - x \tag{3.11}$$

Once all the movements have been performed the feedback() function is executed, and if the real position is different of the setpoint the current position of the joints is updated and the function starts again from the beginning. When the setpoint has been reached it is communicated to the central controller.

3.4.4. Secondary controller requests

The last part of the code is refered to the serial communication with the secondary controller of the robot. The main controller cand send either an order for moving the gripper or a request asking for the real angle of the joints.

When the central controller sends the command for moving the gripper the movegripper condition is turned on, so the main controller sends the same command to the secondary controller, and wait a delay of time til the action has been performed.

On the other hand is the feedback function. It is used when the main controller wants to know the real angle position of each joint. For this it sends the command ‚P‘ through serial communication to the secondary controller, and after waiting a small delay it reads the answer, which contains the angles from the 2nd to the 6th joint, and the 1st joint is measured directly from an analog main controller pin, due to it‘s in the base of the robot.

As happened before the same calculations must be performed for changing the angles from the encoders 5 and 6 to the angles of the joints, but before it must be checked if the abs(motor5)+abs(motor6) is lower than 165º, because as i explained before these SMD encoders has a range of 330º, from -165º to 165º. After it if there is difference between the current position counted by the program and the position of the encoders then the position in the main program is updated.

In this function there is an extra condition, and if the secondary controller didn‘t answer after a delay of 100ms it means that the arduino nano isn‘t connected, so the program keep working without using the feedback() function, trusting in the step counting.

3.5. Programming of the sensors controller

This controller is just waiting for a command from the main controller. When a command is received then it is processed as in the main controller, with the difference that in this case there are just 2 types of possible commands (see **Table 12**).

Table 12. Possible command‘s types to the encoder‘s controller

token[0]	**Action**
‚G‘	Open/Close gripper in a specified percentage
‚P‘	Request for sending the current position

The gripper command is the same as the received in the main controller, but deleting white spaces. An example for opening the gripper a 60% of its maximum is „G60“ (see **Table 13**).

Table 13. Gripper command example 2

pos[0]	**pos[1]**
G	60

Once the string has been divided by tokens the token[1] wich is a percentage value is multiplied by the maximum angle of the servo (180º), and it is sent to the servomotor. The example case is shown from equation 3.12 to equation 3.14.

$$angle_{servo} = angle_{max} \cdot \frac{percentage}{100} \tag{3.12}$$

$$angle_{servo} = 180 \cdot 60/100 = 108º \tag{3.13}$$

$$angle_{servo} = 108º \tag{3.14}$$

In the other hand when asking for the position the received command is ‚P‘. Once it is received the controller just collect the data of the encoders, process them to get numbers and sends them to the main controller.

The data read by the analog pins goes from 0 (0V) to 1023 (5V), so for processing this data this valu is splitted by 1024, multiply it by the angles range of the encoder and substract the half of the range as shown in equation 3.15. Their maximum ranges of measurement are 330º for the SMD encoders and 3600º for the multi-turn encoders.

$$angle = value \cdot \frac{range}{1024} - \frac{range}{2} \tag{3.15}$$

After this calculation the angle received by the SMD encoders will be in the range [-165º,165º], and in the multi-turn encoders [-1800º,1800º], and all these values are sent to the main controller in a string where each angle value is separated by a white space:

„angle2 angle3 angle4 angle5 angle6\n“

3.6. Control app

The GUI has been programmed from python, using the Tkinter library explained in the theoretical annalysis section. This GUI is running in the central controller, and it is accesible from the central screen.
It is divided in 3 main tabs: connection tab, controller tab and path planning tab.

3.6.1. Connection tab

The connection tab is formed by two buttons (see **Fig. 37**), and its purpose is to see if the robots are connected, and if they are not connected you can press the button for trying to perform a connection.

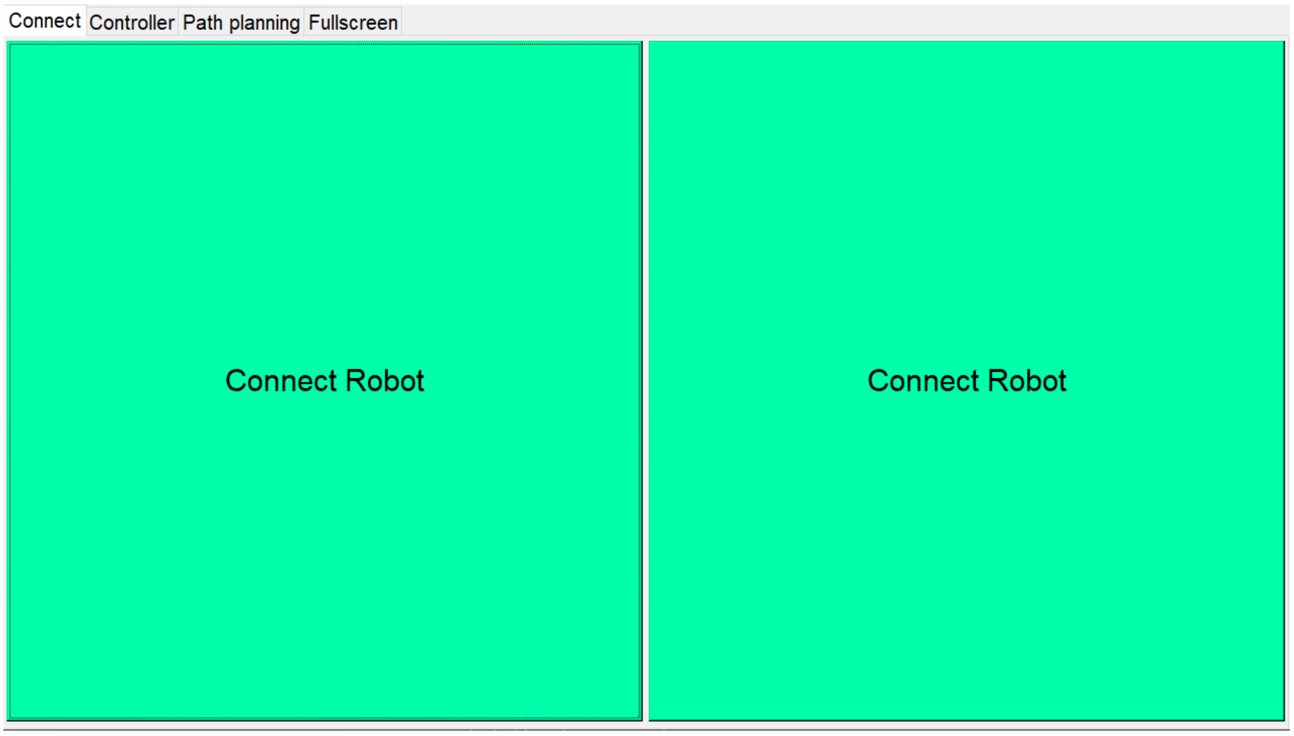


Fig. 37. Application‘s connection tab

When you open the app it tries to perform both connections in the starting. But if the robot is disconnected in the momento f the app starting you always can perform a connection pressing the button. When a robot is connected the button gets disabled and on it is written the robot name:

3.6.2. Controller tab

The controller tab is more complex than the last one, because it allows plenty of actions and different types of control for both arms. And the tab is divided in different parts, numerated in the **Fig. 38**.

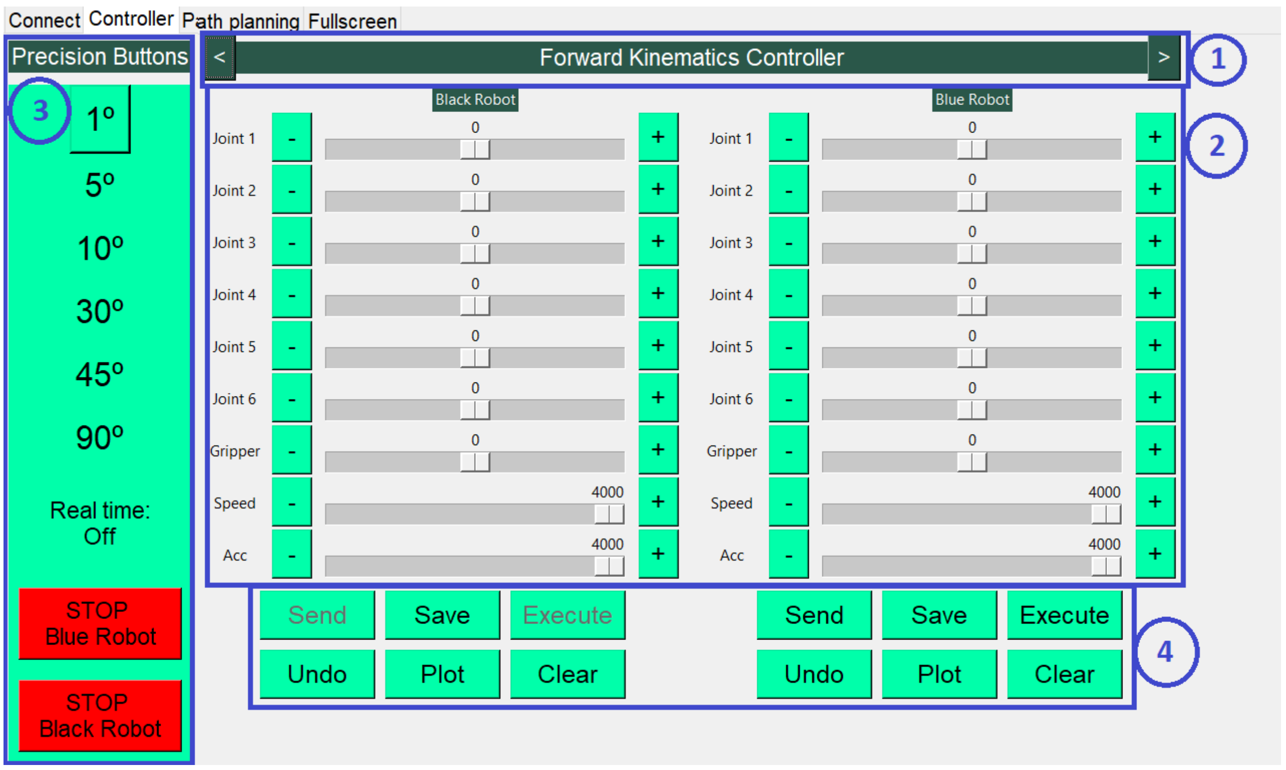


Fig. 38. Application's controller tab

The first part is the type of kinematics applied. In the image it is in forward kinematics mode, what can be appreciated in the box 2, controlling directly the joints angles. Clicking in one of the arrows of the box 1 you can change to inverse kinematics control:

In the box 2 you have the sliders for controlling each one of the arms, in forward or inverse kinematics depending of the box before, and controlling also the gripper percentage of openning and the speed and acceleration of the stepper motors. You can do this by moving the slide or with the buttons next to the sliders. These buttons are a way to move to an exact value in an easier way, and you can change the steps of these ‚+' and ‚-' buttons using the precision buttons of the box 3.

In the box 3 you can choose the value of degrees or milimiters that the buttons next to the sliders will sum up or substract to the sliders value.

Also there are other 3 important buttons in this box. The real time button, that when is activated sends the sliders values to the robot arms in real time, so you can move the robots just moving the sliders. And the others are the stop and continue buttons, which can be pressed in any moment, sending a stop or continue command to the robot, for stop its movement in that position if something goes wrong ori f the movement gets dangerous.

Finally the box 4, which is formed by the actions that can be performed in each arm. In this part the send and execute buttons will only be activated if the arm is connected and waiting for receiving a new command. In the case of this image the blue arm is connected and the black one is not connected. In this box there are six different buttons:

- Send button: This button is used when the real time option is deactivated. Each time this button is pressed the app sends the joints values to the robotic arm.
- Save button: It saves the current sliders values in a list, that will be useful for the execute button or the path planning.
- Execute button: After save different positions using the save button this button will execute of the commands save din the list.
- Undo button: It will update the sliders values to the last command sent to the robot, what means it updates the sliders values to the real position of the robot.
- Plot button: This button plots the current sliders configuration of the robot in a 3D plot, where you can see the robot representation and the end-effector position before of sending it to the robot, as is shown in the **Fig. 39**.
- Clear button: This button eliminates all the commands created by the save button in the execution list.

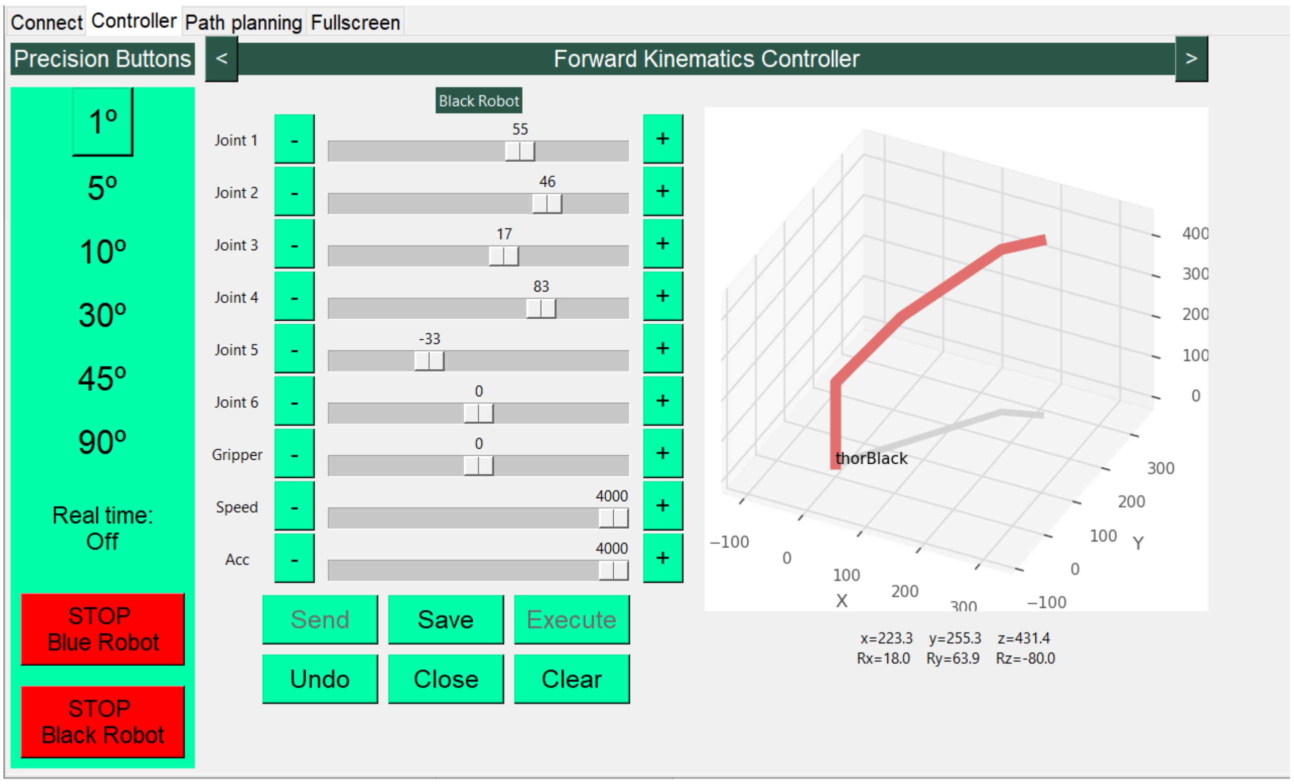


Fig. 39. Plotting option example

3.6.3. Path planning tab

In the path planning tab (see **Fig. 40**) you can create a coordinated path between both robotic arms, that can be created, modified, executed and saved in a file.

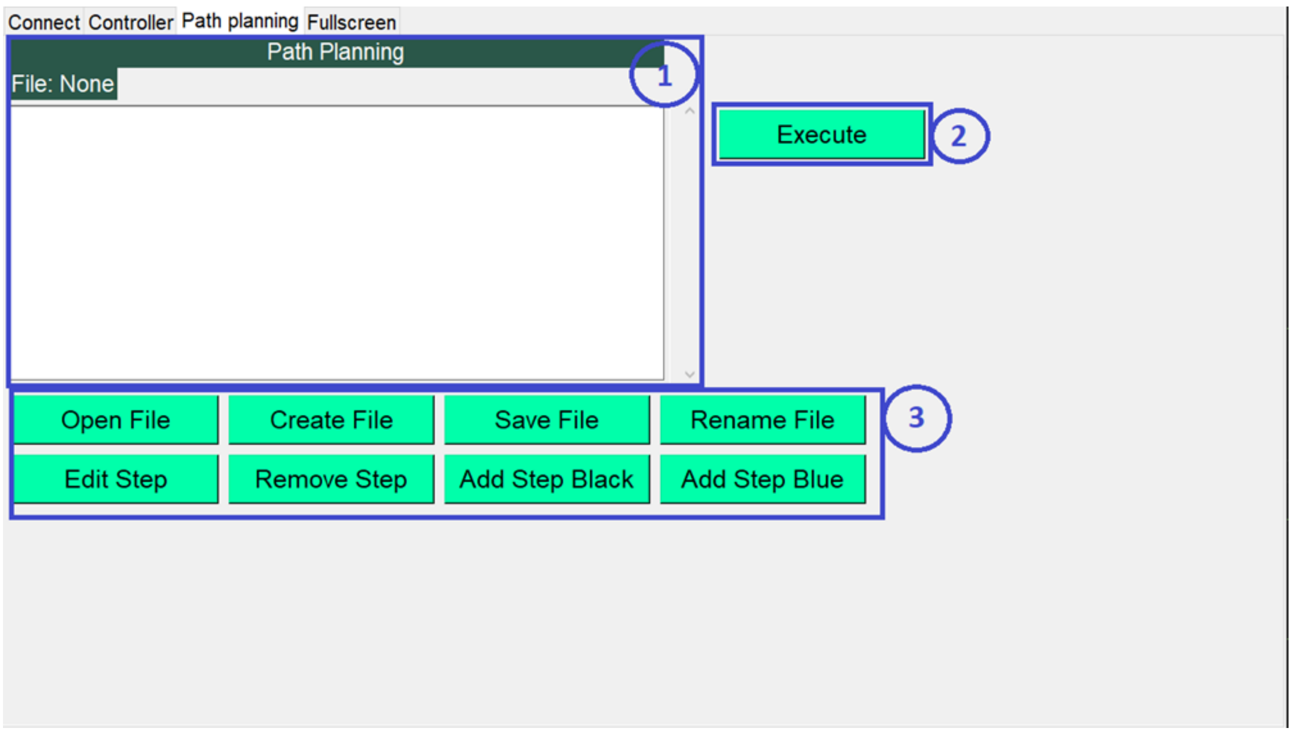
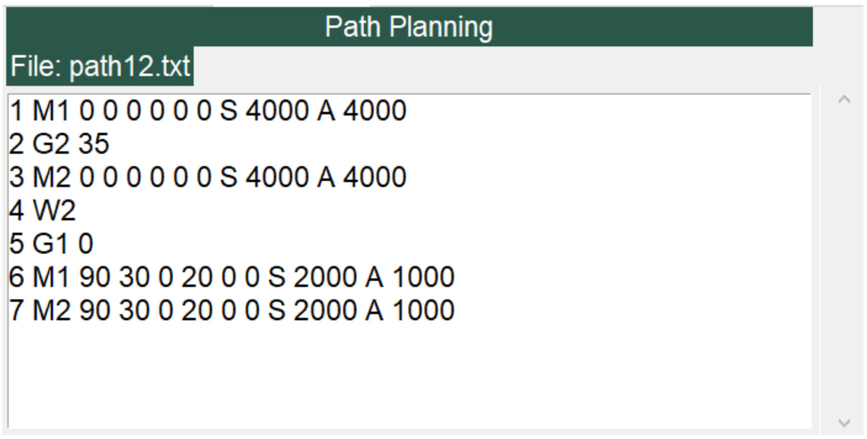


Fig. 40. Application's path planning tab

As is shown in the image there are 3 different boxes. In the first box you can find the name of the file that you are editing or executing, in this case None, because there isn't any file openned, and also you can find a listbox whith all the steps of the path. If you open a file it will appear as shown in **Fig. 41**.



```
1 M1 0 0 0 0 0 0 S 4000 A 4000
2 G2 35
3 M2 0 0 0 0 0 0 S 4000 A 4000
4 W2
5 G1 0
6 M1 90 30 0 20 0 0 S 2000 A 1000
7 M2 90 30 0 20 0 0 S 2000 A 1000
```

Fig. 41. Path example

The commands are the same explained in the programming of the robot controller, adding the white command and a number next to the command letter.

The first letter of the command means what type of command it is, as shown in **Table 14**.

Table 14. Possible commands of the path creator

token[0]	**Action**
‚M‘	Move joints to a new configuration of angles
‚G‘	Open/Close gripper in a specified percentage
‚W‘	Program waits til the defined arm finishes the movement

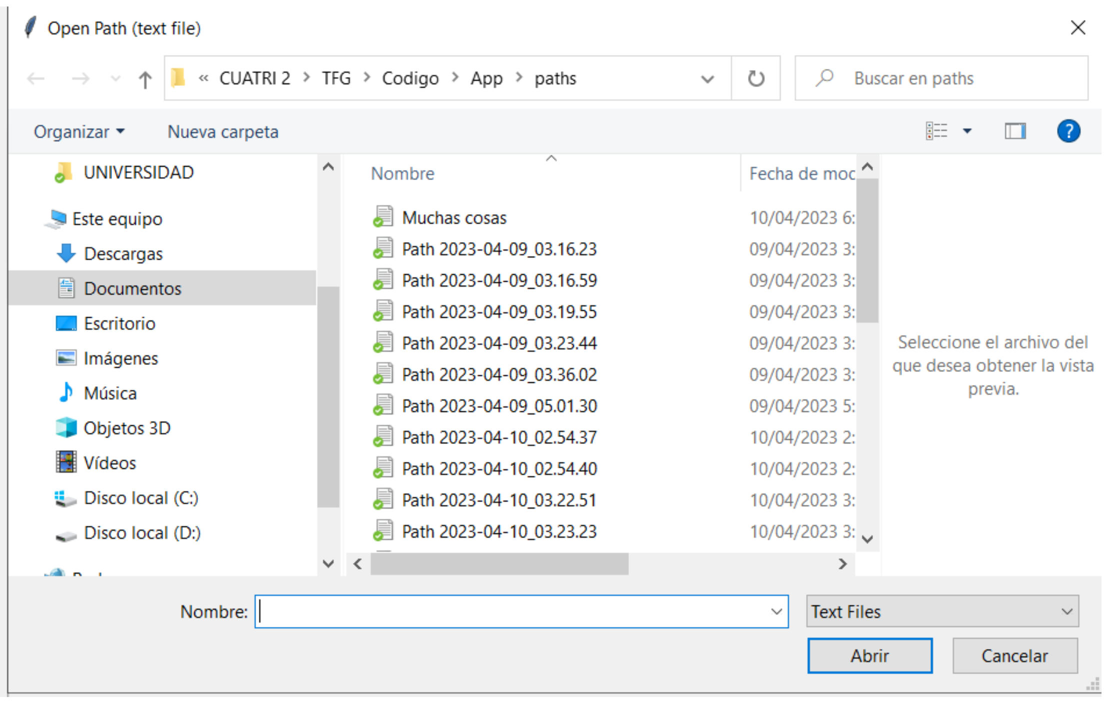
Each of these commands has a number next to them, which means in what robot will be applied (see **Table 15**).

Table 15. Robot‘s number for the commands

Number	**Robot**
1	Black
2	Blue

Then this path would:
1. Move black robot to angles {0,0,0,0,0,0} in a max speed of 4000 steps/s and acceleration of 4000steps/s^2.
2. Open the gripper of the blue robot to the 35%, while black robot is moving, because there isn‘t any wait command.
3. When the blue robot finished its action it will moves to angles {0,0,0,0,0,0} in a max speed of 4000 steps/s and acceleration of 4000steps/s^2.
4. Wait until the blue robot finishes the movement.
5. Close the gripper of the black robot.
6. Move black robot to angles {90,30,0,20,0,0} in a max speed of 2000 steps/s and acceleration of 1000 steps/s^2.
7. Move blue robot to angles {90,30,0,20,0,0} in a max speed of 2000 steps/s and acceleration of 1000 steps/s^2 while black robot is still moving, because there isn‘t a wait command.

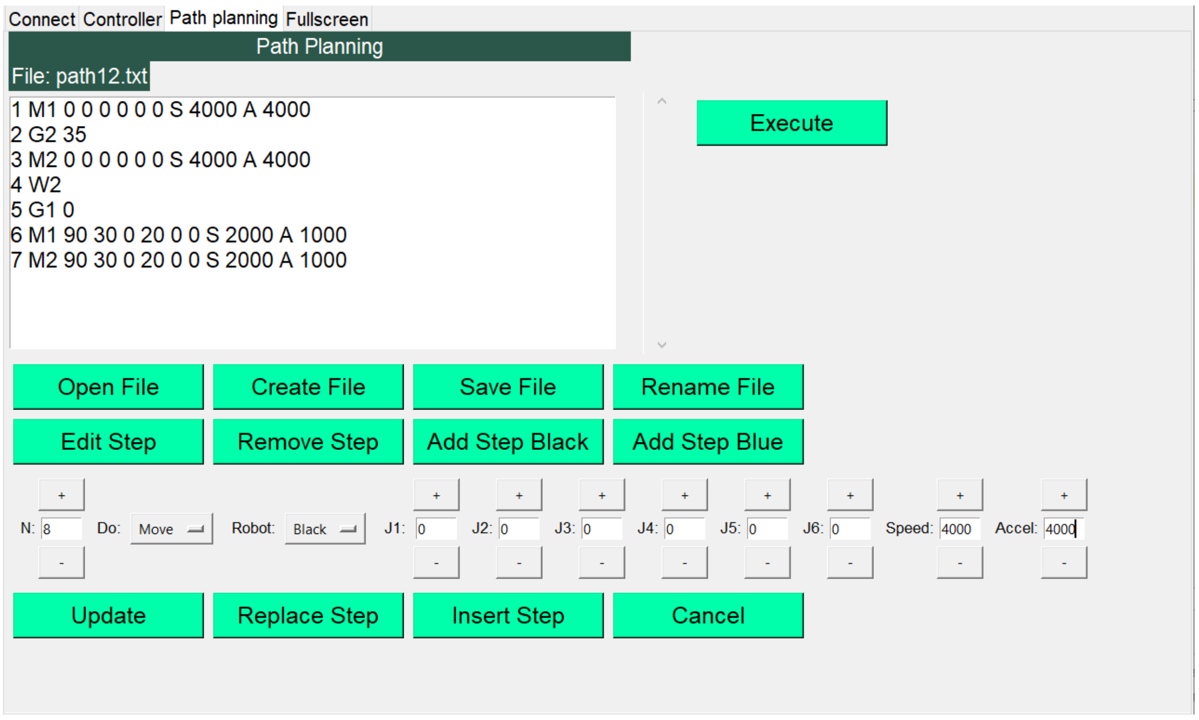
In the second box you have the execute button, whose function is obviously to execute the path, and when it is pressed it is changed by other two buttons in its place, „Stop button“ and „Finish button“.
Lastly it is the third box, formed by the buttons with all the options implemented for creating, editing and saving paths in permanent files.
The first row of buttons is easy to understand, you can open, create, save and rename files where the paths will be allocated. These files are in a folder named „paths“, in the same directory as the application. When you press the button „Open file“ a new window is open, which you can use for searching paths in this directory, as shown in **Fig. 42**.



Nombre	Fecha de moc
Muchas cosas	10/04/2023 6:
Path 2023-04-09_03.16.23	09/04/2023 3:
Path 2023-04-09_03.16.59	09/04/2023 3:
Path 2023-04-09_03.19.55	09/04/2023 3:
Path 2023-04-09_03.23.44	09/04/2023 3:
Path 2023-04-09_03.36.02	09/04/2023 3:
Path 2023-04-09_05.01.30	09/04/2023 5:
Path 2023-04-10_02.54.37	10/04/2023 2:
Path 2023-04-10_02.54.40	10/04/2023 2:
Path 2023-04-10_03.22.51	10/04/2023 3:
Path 2023-04-10_03.23.23	10/04/2023 3:

Fig. 42. Menu for openning files

As you can appreciate in the last image each time you create a new file from the application if you don‘t rename it the file will be created under the name: „Path date_hour“.
Apart from control the files from the application they can be edit too, using the others buttons. If you press the buttons „Add Step Black“ or „Add Step Blue“ a new frame will appear in the tab (see **Fig. 43**), where you can create the command that you want to add.



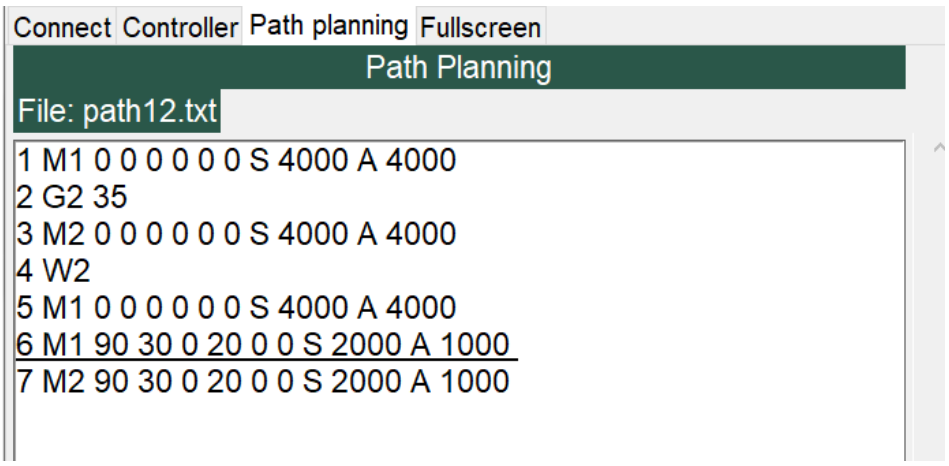
File: path12.txt

```
1 M1 0 0 0 0 0 0 S 4000 A 4000
2 G2 35
3 M2 0 0 0 0 0 0 S 4000 A 4000
4 W2
5 G1 0
6 M1 90 30 0 20 0 0 S 2000 A 1000
7 M2 90 30 0 20 0 0 S 2000 A 1000
```

N: 8 Do: Move Robot: Black J1: 0 J2: 0 J3: 0 J4: 0 J5: 0 J6: 0 Speed: 4000 Accel: 4000

Fig. 43. Menu for adding steps in path‘s creator tab

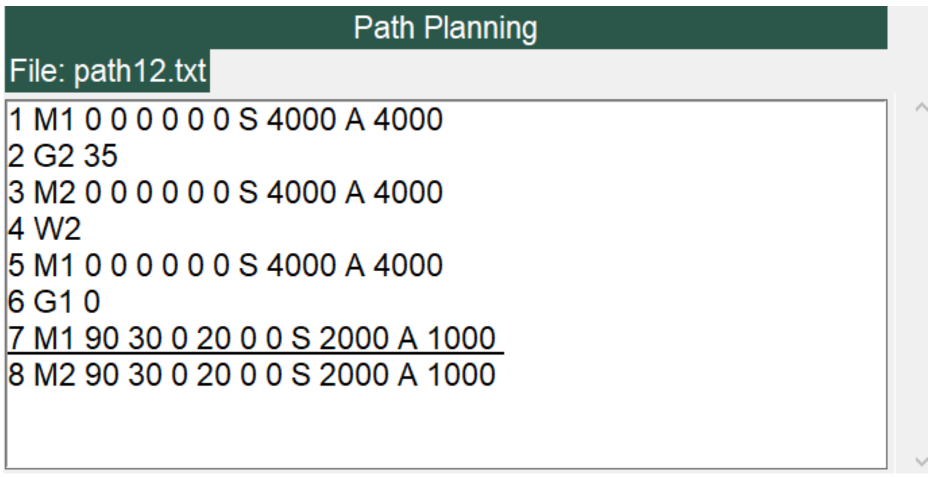
As is shown in this new menu it can be chosen the step number of the command, the action, the robot to be applied, and next to it boxes with each value of the command. When the add button is pressed it automatically takes the last values sent to the robot or saved in the controller tab. They can be edited also from this tab, or even edit them in the controller tab and press the save button, so coming back to this tab and press update the values will be updated.
Once the new command has been defined you have two ways to add it to the path, replace or insert. If the step number chosen is the next one both buttons will insert the step into the path without difference, as in the case of the image, because the step number chosen is 8, and the path has 7 steps. However it can be added as the fifth step, and there are two different ways of doing it.
On the one hand using the replace button, so the fifth step will be eliminated and replaced by this new command as shown in **Fig. 44**.



```
1 M1 0 0 0 0 0 0 S 4000 A 4000
2 G2 35
3 M2 0 0 0 0 0 0 S 4000 A 4000
4 W2
5 M1 0 0 0 0 0 0 S 4000 A 4000
6 M1 90 30 0 20 0 0 S 2000 A 1000
7 M2 90 30 0 20 0 0 S 2000 A 1000
```

Fig. 44. Path example with step replacement

On the other hand using the insert button. In this case the steps from the fifth to the last will be displaced by 1 and the new command will be inserted in the fifth position (see **Fig. 45**).



```
1 M1 0 0 0 0 0 0 S 4000 A 4000
2 G2 35
3 M2 0 0 0 0 0 0 S 4000 A 4000
4 W2
5 M1 0 0 0 0 0 0 S 4000 A 4000
6 G1 0
7 M1 90 30 0 20 0 0 S 2000 A 1000
8 M2 90 30 0 20 0 0 S 2000 A 1000
```

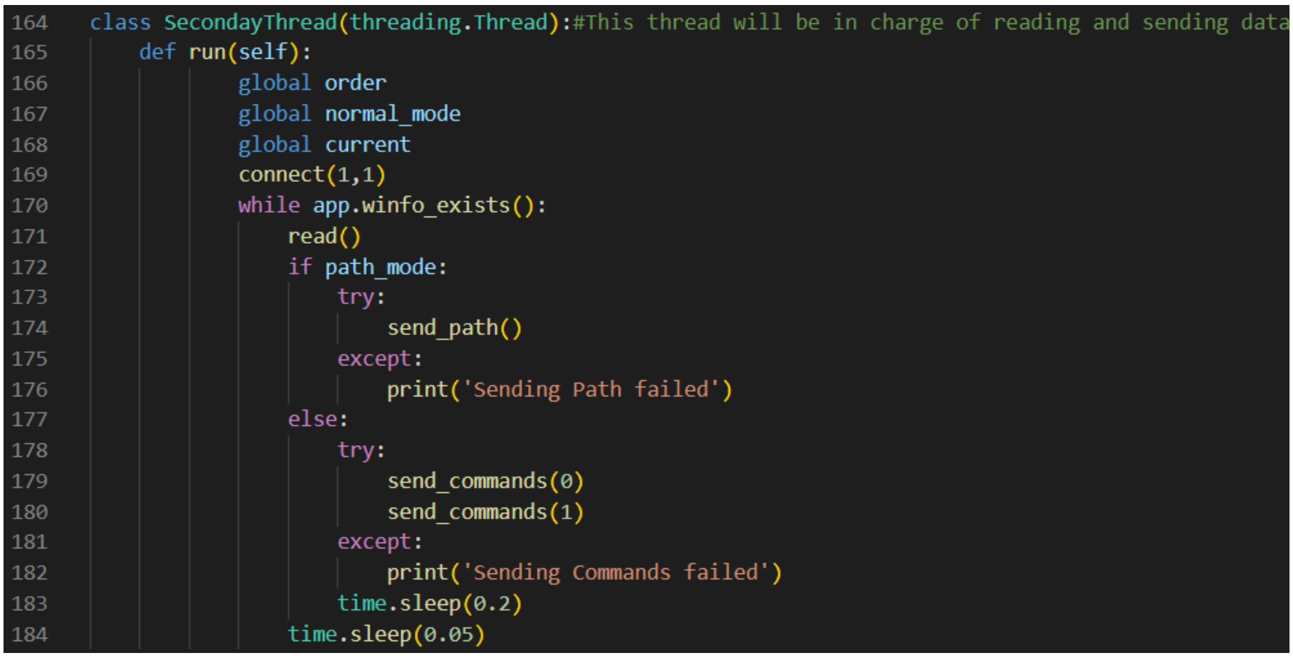
Fig. 45. Path example with step inserted

Furthermore steps already added can be editted or removed using the two remaining buttons.

3.7. Control app backend

3.7.1. Serial communication

In the case of the app the serial communication between the central controller and the robots controller is managed in a second thread (see **Fig. 46**). This second thread is in charge of perform the connections, read the serial port and send the commands to the robots.



```
class SecondayThread(threading.Thread):#This thread will be in charge of reading and sending data
    def run(self):
        global order
        global normal_mode
        global current
        connect(1,1)
        while app.winfo_exists():
            read()
            if path_mode:
                try:
                    send_path()
                except:
                    print('Sending Path failed')
            else:
                try:
                    send_commands(0)
                    send_commands(1)
                except:
                    print('Sending Commands failed')
                time.sleep(0.2)
            time.sleep(0.05)
```

Fig. 46. Second thread for serial communication

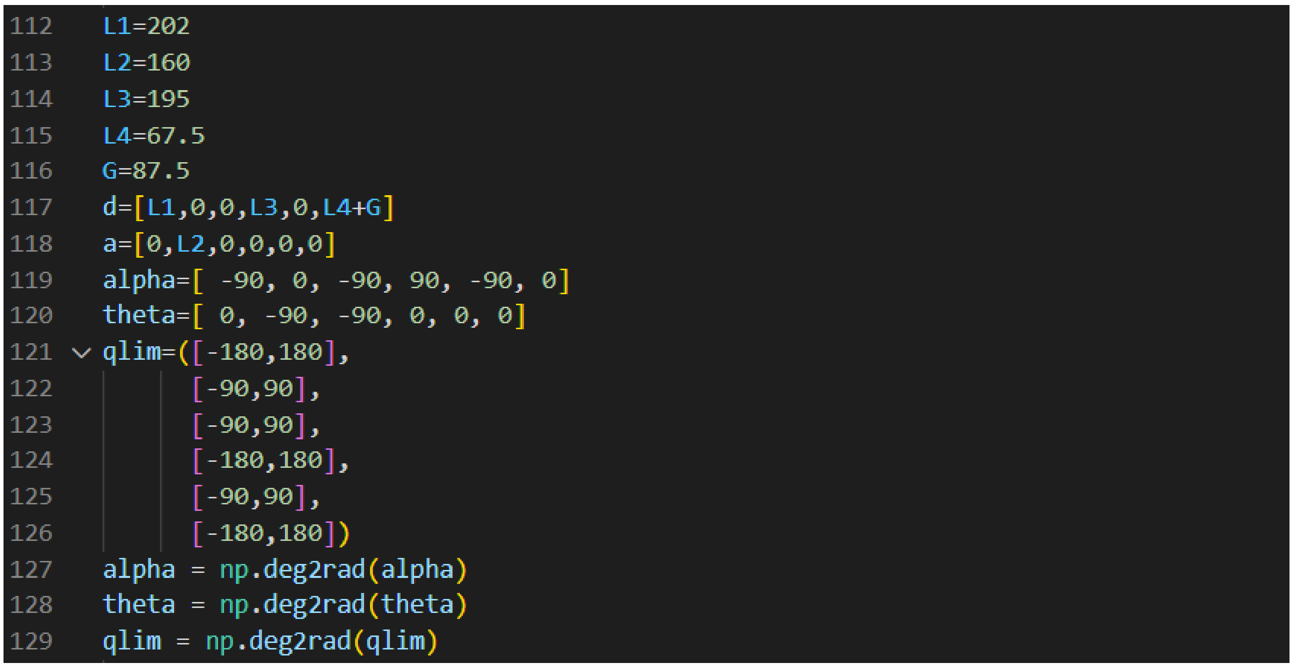
The connect(bool robot1,bool robot2) function is used in the starting of the second thread, for connecting both robots in the starting of the app. After that the thread enters in an infinite loop where it will send commands to the robots and receiving status messages from them.

For send commands the thread has two modes: path_mode or normal_mode. Depending on this condition the app will execute the function send_path() or the functions for sending commands to the robots when needed.

Either way the app will read at each iteration if there are any status messages from the arms.

3.7.2. Robotics toolbox

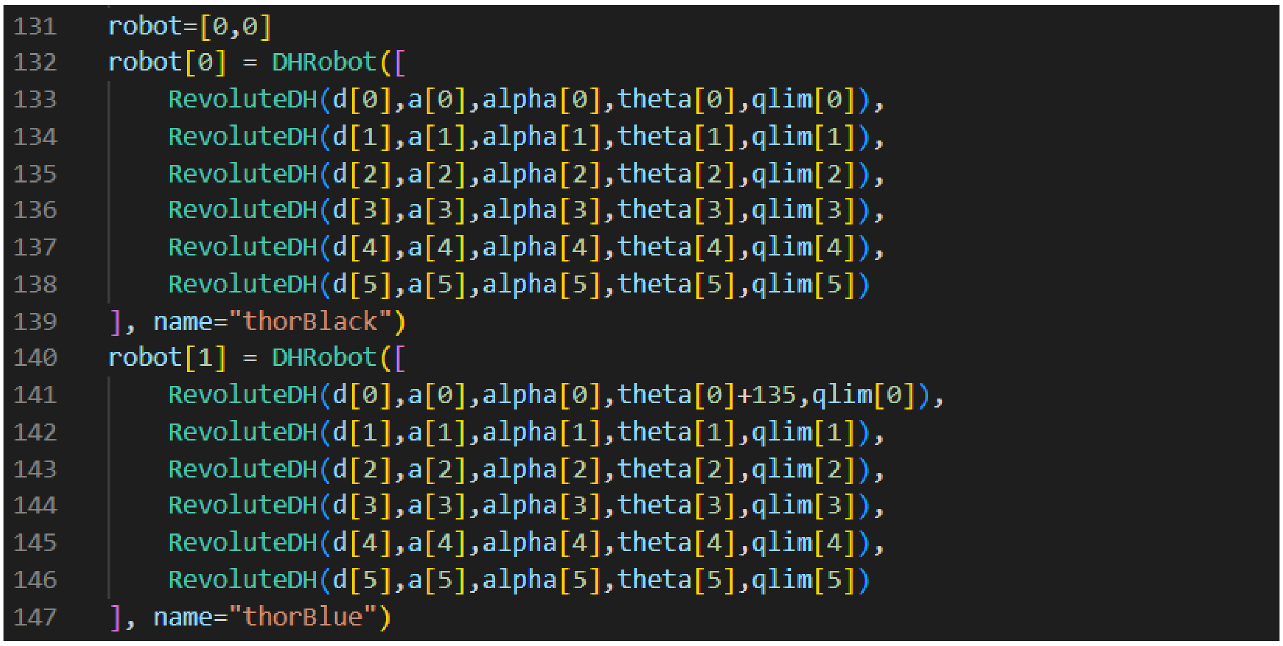
For creating some functions of the app, and making possible other future applications has been used the robotics toolbox for python explained in the theoretical analysis. Thanks to this toolbox it has been modeled the robots in the application, and functions for inverse and forward kinematic and for plotting robots can be used.
Robots have been modeled using the D-H table showed before, and the limit angles of each joint (see **Fig. 47**).



```python
L1=202
L2=160
L3=195
L4=67.5
G=87.5
d=[L1,0,0,L3,0,L4+G]
a=[0,L2,0,0,0,0]
alpha=[ -90, 0, -90, 90, -90, 0]
theta=[ 0, -90, -90, 0, 0, 0]
qlim=([-180,180],
        [-90,90],
        [-90,90],
        [-180,180],
        [-90,90],
        [-180,180])
alpha = np.deg2rad(alpha)
theta = np.deg2rad(theta)
qlim = np.deg2rad(qlim)
```

Fig. 47. Robot parameters

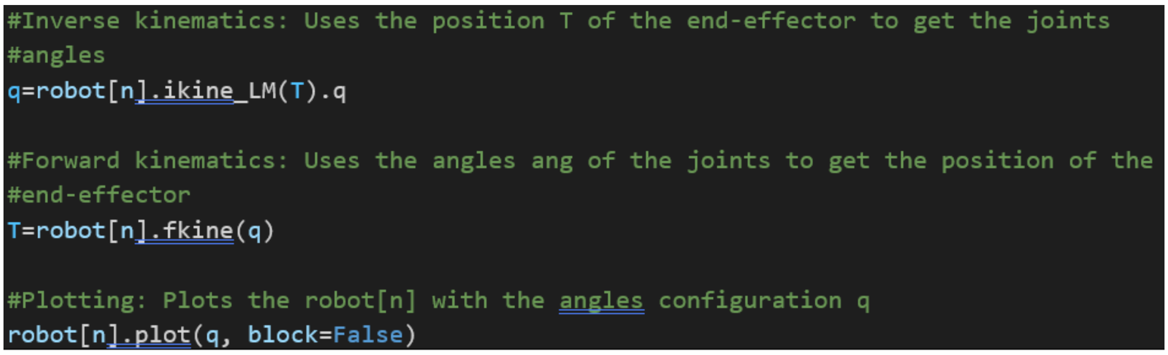
Using these values it has been created the robots using the DHRobot function from this library as shown in **Fig. 48**.



```python
robot=[0,0]
robot[0] = DHRobot([
    RevoluteDH(d[0],a[0],alpha[0],theta[0],qlim[0]),
    RevoluteDH(d[1],a[1],alpha[1],theta[1],qlim[1]),
    RevoluteDH(d[2],a[2],alpha[2],theta[2],qlim[2]),
    RevoluteDH(d[3],a[3],alpha[3],theta[3],qlim[3]),
    RevoluteDH(d[4],a[4],alpha[4],theta[4],qlim[4]),
    RevoluteDH(d[5],a[5],alpha[5],theta[5],qlim[5])
], name="thorBlack")
robot[1] = DHRobot([
    RevoluteDH(d[0],a[0],alpha[0],theta[0]+135,qlim[0]),
    RevoluteDH(d[1],a[1],alpha[1],theta[1],qlim[1]),
    RevoluteDH(d[2],a[2],alpha[2],theta[2],qlim[2]),
    RevoluteDH(d[3],a[3],alpha[3],theta[3],qlim[3]),
    RevoluteDH(d[4],a[4],alpha[4],theta[4],qlim[4]),
    RevoluteDH(d[5],a[5],alpha[5],theta[5],qlim[5])
], name="thorBlue")
```

Fig. 48. Robots declaration

Once they are created its current position can be plotted, update it, and realize calculations for forward and inverse kinematics. In the case of the inverse kinematics is used the function with the algorithm of Levenberg-Marquardt. The main functions of this library used in the application are the written in the **Fig. 49**.



```
#Inverse kinematics: Uses the position T of the end-effector to get the joints
#angles
q=robot[n].ikine_LM(T).q

#Forward kinematics: Uses the angles ang of the joints to get the position of the
#end-effector
T=robot[n].fkine(q)

#Plotting: Plots the robot[n] with the angles configuration q
robot[n].plot(q, block=False)
```

Fig. 49. Robotics toolbox functions used in the backend

3.8. Implementation of wireless screen

The wireless screen has been added to the raspberry using teamviewer. For avoiding the delay the control has been performed in a local red, created in the raspberry pi, and a static ip has been chosen for the raspberry pi: 10.42.0.1

After it teamviewer has been installed in both, raspberry pi and tablet, and configured for accepting just local connections, and in the raspberry pi a permanent password has been chosen for remote connections.

Once these configurations has been finished the tablet can already perform a stable connection with the raspberry pi following the next step:

- Connect tablet to the Raspberry's hotspot.
- Open teamviewer and introduce the static ip of the raspberry pi.
- Enter the chosen password.

Following these simple steps you can already control the robotic station wireless (see **Fig. 50**).

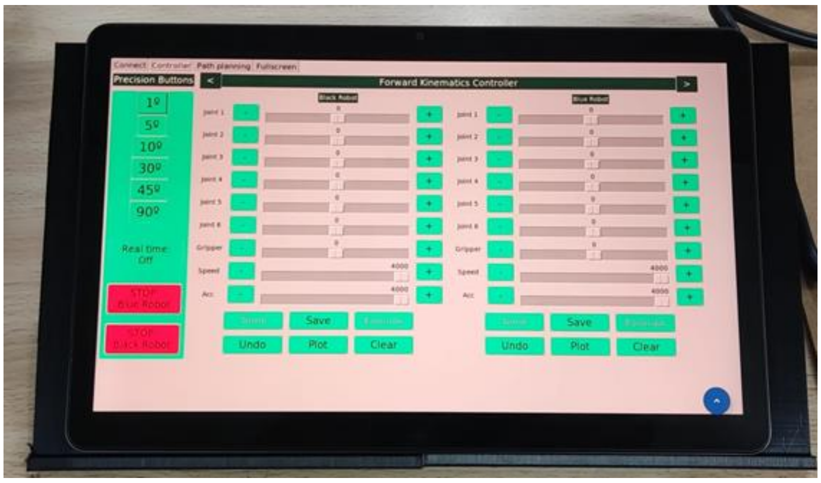


Fig. 50. Robotic Station wireless monitor

Results and conclusions

1. After researching about the different types of open-source robotic arms it has been decided to use the Thor model with addons, as it already incorporates support for feedback sensors in the design, and it is a 6 DOF antropomorphic arm with a high payload, taking into account that it is fully printed in 3D (up to 0.75 kg) and the highest range of action of the studied 3D printed arms, with a reach of 598 mm.
2. The electrical and electronic componentes of the robotic station have been selected and tested based on system needings, and the designed electrical and control schematics works as expected. Every part of the robotic station is correctly powered and the control of both robots has been successful.
3. Both robotic arms has been assemblied and wired. Both arms can move all the joints throughout its entire range of motion, as exepected in the simulations with the virtual model created in RoboDK.
4. Both arms has been assemblied in the same table, and has been connected to the central controller and the general power of the robotic station, achieving to work together simultaneously without issues in communications or power supply.
5. Each robot controller has been programmed for realising the actions sent by the central controller to the the main controller of each arm, as moving the gripper, stop, and move the joints to a designed position, with the capacity of controlling all the motors at the same time, counting steps and realising a feedback periodically. The gripper actions and the encoders data collection is performed by the secondary controller of the arm, connected to the main controller by serial communication.
6. A wireless touch screen has been implemented in the robotic station, in order to control the central computer GUI. The connection is stable and according to the tests there is no appreciable delay in a range of 10 meters.
7. A GUI has been created from scratch to control the robotic station. This GUI enables a wide range of functionalities, including establishing connections with the robots, forward and inverse kinematics controlling, possibility of using real-time control, stoping and continuing its movements instantaneously and creating of coordinated paths for both arms, which can be executed by the robots or saved in a file.

List of references

[1] M. E. Moran, “Evolution of robotic arms,” *J Robot Surg*, vol. 1, no. 2, pp. 103–111, May 2007, doi: 10.1007/S11701-006-0002-X/FIGURES/12.

[2] “Robotic Arms: A Brief - The ERC Blog.” https://erc-bpgc.github.io/blog/blog/robotic_arms/ (accessed May 14, 2023).

[3] “Fanuc launches new 1,000 kg payload industrial robotic arm.” https://roboticsandautomationnews.com/2022/01/24/fanuc-launches-new-1000-kg-payload-industrial-robotic-arm/48552/ (accessed May 14, 2023).

[4] O. Stan and A. Manea, “Design and Implement a 6DOF Anthropomorphic Robotic Structure,” *International Journal of Modeling and Optimization*, pp. 352–356, Dec. 2019, doi: 10.7763/IJMO.2019.V9.736.

[5] C. Smith *et al.*, “Dual arm manipulation - A survey,” *Rob Auton Syst*, vol. 60, no. 10, pp. 1340–1353, 2012, doi: 10.1016/J.ROBOT.2012.07.005.

[6] “ABB introduces YuMi®, world’s first truly collaborative dual-arm robot.” https://new.abb.com/news/detail/12952/abb-introduces-yumir-worlds-first-truly-collaborative-dual-arm-robot (accessed May 15, 2023).

[7] J. Krüger, G. Schreck, and D. Surdilovic, “Dual arm robot for flexible and cooperative assembly,” *CIRP Ann Manuf Technol*, vol. 60, no. 1, pp. 5–8, 2011, doi: 10.1016/J.CIRP.2011.03.017.

[8] B. Sheng, W. Meng, C. Deng, and S. Xie, “Model based kinematic & dynamic simulation of 6-DOF upper-limb rehabilitation robot,” *Proceedings of 2016 Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2016*, pp. 21–25, Aug. 2016, doi: 10.1109/ACIRS.2016.7556181.

[9] F. Piltan, A. Taghizadegan, and N. B. Sulaiman, “Modeling and Control of Four Degrees of Freedom Surgical Robot Manipulator Using MATLAB/SIMULINK,” *International Journal of Hybrid Information Technology*, vol. 8, no. 11, pp. 47–78, Nov. 2015, doi: 10.14257/IJHIT.2015.8.11.05.

[10] A. A. Goldenberg and R. G. Fenton, “A Complete Generalized Solution to the Inverse Kinematics of Robots,” *IEEE JOURNAL OF ROBOTICS AND AUTOMATION*, vol. 1, no. 1, 1985.

[11] G. Luo, L. Zou, Z. Wang, C. Lv, J. Ou, and Y. Huang, “A novel kinematic parameters calibration method for industrial robot based on Levenberg-Marquardt and Differential Evolution hybrid algorithm,” *Robot Comput Integr Manuf*, vol. 71, p. 102165, Oct. 2021, doi: 10.1016/J.RCIM.2021.102165.

[12] T. Sugihara, “Solvability-unconcerned inverse kinematics by the Levenberg-Marquardt method,” *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 984–991, Oct. 2011, doi: 10.1109/TRO.2011.2148230.

[13] “What is an Open Loop System? Examples, Advantages, Disadvantages.” https://www.electronicshub.org/open-loop-system/ (accessed May 15, 2023).

[14] “Closed Loop Control System : Block Diagram, Types & Its Applications.” https://www.elprocus.com/what-is-a-closed-loop-control-system-its-working/ (accessed May 15, 2023).

[15] K. Rahul, H. Raheman, and V. Paradkar, “Design of a 4 DOF parallel robot arm and the firmware implementation on embedded system to transplant pot seedlings,” *Artificial Intelligence in Agriculture*, vol. 4, pp. 172–183, Jan. 2020, doi: 10.1016/J.AIIA.2020.09.003.

[16] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, H. V. Rios-Figueroa, H. Vazquez-Leal, and E. R. Palacios-Hernandez, “Design and implementation of a robotic arm using ROS and MoveIt!,” *2017 IEEE International Autumn Meeting on Power, Electronics and Computing, ROPEC 2017*, vol. 2018-January, pp. 1–6, Jul. 2017, doi: 10.1109/ROPEC.2017.8261666.

[17] P. Corke and J. Haviland, “Not your grandmother’s toolbox - the Robotics Toolbox reinvented for Python,” *Proc IEEE Int Conf Robot Autom*, vol. 2021-May, pp. 11357–11363, 2021, doi: 10.1109/ICRA48506.2021.9561366.

[18] “C++ GUI Programming with Qt 4 - Jasmin Blanchette, Mark Summerfield - Google Libros.” https://books.google.es/books?hl=es&lr=&id=tSCR_4LH2KsC&oi=fnd&pg=PR5&dq=c%2B%2B+qt+gui&ots=Ea0YlW23Ah&sig=lx7V9UZtHRWqr9Vv-oHiIhOVFnY#v=onepage&q=c%2B%2B%20qt%20gui&f=false (accessed Apr. 25, 2023).

[19] G. Polo, “PyGTK, PyQT, Tkinter and wxPython comparison”, Accessed: Apr. 24, 2023. [Online]. Available: http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/XHI

[20] Y. Güven, E. Coşgun, S. Kocaoğlu, H. Gezici, and E. Yılmazlar, “Understanding the concept of microcontroller based systems to choose the best hardware for applications.,” *Research Inventy: International Journal of Engineering And Science*, vol. 6, no. 9, pp. 2319–6483, Jul. 2019, Accessed: Apr. 24, 2023. [Online]. Available: http://acikerisim.klu.edu.tr/xmlui/handle/20.500.11857/1024

[21] H. Kareem and D. Dunaev, “The Working Principles of ESP32 and Analytical Comparison of using Low-Cost Microcontroller Modules in Embedded Systems Design,” *2021 4th International Conference on Circuits, Systems and Simulation, ICCSS 2021*, pp. 130–135, May 2021, doi: 10.1109/ICCSS51193.2021.9464217.

[22] “RaspberryPI models comparison | Comparison tables - SocialCompare.” https://socialcompare.com/es/comparison/raspberrypi-models-comparison (accessed May 15, 2023).

[23] A. Tai, M. Chun, Y. Gan, M. Selamet, and H. Lipson, “PARA: A one-meter reach, two-kg payload, three-DoF open source robotic arm with customizable end effector,” *HardwareX*, vol. 10, Oct. 2021, doi: 10.1016/J.OHX.2021.E00209.

[24] “moveo brazo robótico BCN3D Open Source | Sicnova.” https://sicnova3d.com/blog/moveo-el-brazo-robotico-open-source-de-bcn3d/ (accessed May 15, 2023).

[25] “6DoF Robot Arm (Six-Axis 3D printed Robotic Arm) por Skyentific.” https://www.printables.com/es/model/5311-6dof-robot-arm-six-axis-3d-printed-robotic-arm (accessed May 15, 2023).

[26] J. Junia Santillo Costa, M. Lajovic Carneiro, and T. Antonio Urzedo Machado, “Implementation and Validation of Thor 3D Printed Open Source Robotic Arm,” *IEEE Latin America Transactions*, vol. 18, no. 5, pp. 907–913, May 2020, doi: 10.1109/TLA.2020.9082919.

[27] “Thor – An Open Source 3D Printable 6DOF Robotic Arm.” http://thor.angel-lm.com/ (accessed May 15, 2023).

[28] “‘Thor’ robot with addons and GUI | Hackaday.io.” https://hackaday.io/project/16665-thor-robot-with-addons-and-gui (accessed May 15, 2023).

[29] “Forward Kinematics | Details | Hackaday.io.” https://hackaday.io/project/12989-thor/log/43941-forward-kinematics (accessed May 15, 2023).

[30] P. Sivasankaran* and R. Karthikeyan, “Simulation of Robot Kinematic Motions using Collision Mapping Planner using Robo Dk Solver,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 11, pp. 21–27, Sep. 2020, doi: 10.35940/IJITEE.J7588.0991120.

[31] P. Chand, “Developing a Matlab Controller for Niryo Ned Robot,” *Proceedings - 2022 1st International Conference on Technology Innovation and Its Applications, ICTIIA 2022*, 2022, doi: 10.1109/ICTIIA54654.2022.9935911.

[32] Y. Shin, S. Seol, and K. Lee, “A study on quality of experience of controlling a device remotely in an IoT environment,” *International Conference on Ubiquitous and Future Networks, ICUFN*, vol. 2016-August, pp. 699–702, Aug. 2016, doi: 10.1109/ICUFN.2016.7537126.

[33] “A4988-DMOS Microstepping Driver with Translator And Overcurrent Protection”, Accessed: May 11, 2023. [Online]. Available: www.allegromicro.com

Appendices

Appendix 1. Central controller code.

```
#!/usr/bin/env python
#Pendiente hacer funcionar las inverse/forward kinematics
import tkinter as tk
import datetime
import os
from tkinter import ttk
from tkinter import filedialog
import roboticstoolbox as rtb
from roboticstoolbox import DHRobot, RevoluteDH
import numpy as np
import threading
import time
import serial as se
import serial.tools.list_ports as ser_tool
from spatialmath.base import eul2tr
from spatialmath import SE3
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
#Connection variables---------------------------------------------------------
---------------------
robot_a=-1
robot_b=-1
serial=[0,0]
connected=[0,0]
ready=[0,0]
ports=[-1,-1]

#Communication variables------------------------------------------------------------
------------------------
real_time=0
stopped=[0,0]
stop=[0,0]

#Normal mode variables--------------------------------------------------------------
----------------------
normal_mode=[1,1]
order=0
current=0
orders=[]

#Execution mode variables-----------------------------------------------------------
-------------------------
execution_mode=[0,0]
exec_order=[0,0]
exec_current=[0,0]
```

```
exec_list_w=[]
exec_list_b=[]
exec_list=[exec_list_w,exec_list_b]
exec_list.append(exec_list_w)
exec_list.append(exec_list_b)

#Robots values variables-----------------------------------------------------
-------------------------
current_values=[[0,0,0,0,0,0,0,4000,4000],[0,0,0,0,0,0,0,4000,4000]]
last_saved=[[0,0,0,0,0,0,0,4000,4000],[0,0,0,0,0,0,0,4000,4000]]

#Control variables-----------------------------------------------------------
------------------
ik=0
plot_mode=0
total_change=0
list_accuracy=[1, 5, 10, 30, 45, 90]
list_accuracy_mm=[1, 5, 10, 20, 50, 100]
list_joints=['Joint 1','Joint 2','Joint 3','Joint 4','Joint 5','Joint 6','Gripper',
'Speed', 'Acc']
list_axis=['x','y','z','Rx','Ry','Rz','Gripper', 'Speed', 'Acc']
axis_joints_min=[-450,-450,0,-360,-360,-360,-90,0,0]
axis_joints_max=[450,450,800,360,360,360,90,4000,4000]
angle_joints_min=[-180,-90,-90,-180,-90,-180,0,0,0]
angle_joints_max=[180,90,90,180,90,180,100,4000,4000]
current_sliders=[[0,0,0,0,0,0],[0,0,0,0,0,0]]
current_sliders_ik=[[0,0,0,0,0,0],[0,0,0,0,0,0]]
accuracy=1
l=len(list_joints)

#Initialize variables for Tkinter Control components-------------------------------
----------------------------------------------------
color1='#00FFAA' #Current: '#00FFAA'
color2='#2A5749' #Current: '#2A5749'
precis_button=[]
stop_button=[]
robots_labels=['Black Robot', 'Blue Robot']
stop_labels=['STOP \nBlack Robot','STOP \nBlue Robot','CONTINUE \nBlack Robot',
'CONTINUE \nBlue Robot']
joint_slider=[[],[]]
joint_label=[[],[]]
minus_button=[[],[]]
plus_button=[[],[]]
control_frame_robots=[None,None]
control_frame_buttons=[None,None]
undo_button=[None,None]
send_button=[None,None]
plot_button=[None,None]
save_button=[None,None]
execute_button=[None,None]
clear_button=[None,None]
```

```
r_lbl=[None,None]
r_lbl_up=[None,None]

#Path variables------------------------------------------------------------------
---------------
wait=[0,0]
path_stopped=0
path_stop=0
path_i=0
path_size=0
path_mode=0
edit_item=0
step_robot_option=0
path_text=""
current_path=[]
text_file_dir=""
step_complements_lbl=[0,0,0,0,0,0,0,0]
step_comp_sel=[0,0,0,0,0,0,0,0]
step_comp_plus=[0,0,0,0,0,0,0,0]
step_comp_minus=[0,0,0,0,0,0,0,0]

#Robots variables:---------------------------------------------------------------
------------------
    #Geometry
L1=202
L2=160
L3=195
L4=67.5
G=87.5
d=[L1,0,0,L3,0,L4+G]
a=[0,L2,0,0,0,0]
alpha=[ -90, 0, -90, 90, -90, 0]
theta=[ 0, -90, -90, 0, 0, 0]
qlim=([-180,180],
      [-90,90],
      [-90,90],
      [-180,180],
      [-90,90],
      [-180,180])
alpha = np.deg2rad(alpha)
theta = np.deg2rad(theta)
qlim = np.deg2rad(qlim)
    #Robot declaration
robot=[0,0]
robot[0] = DHRobot([
    RevoluteDH(d[0],a[0],alpha[0],theta[0],qlim[0]),
    RevoluteDH(d[1],a[1],alpha[1],theta[1],qlim[1]),
    RevoluteDH(d[2],a[2],alpha[2],theta[2],qlim[2]),
    RevoluteDH(d[3],a[3],alpha[3],theta[3],qlim[3]),
    RevoluteDH(d[4],a[4],alpha[4],theta[4],qlim[4]),
    RevoluteDH(d[5],a[5],alpha[5],theta[5],qlim[5])
], name="thorBlack")
robot[1] = DHRobot([
    RevoluteDH(d[0],a[0],alpha[0],theta[0]+135,qlim[0]),
    RevoluteDH(d[1],a[1],alpha[1],theta[1],qlim[1]),
    RevoluteDH(d[2],a[2],alpha[2],theta[2],qlim[2]),
    RevoluteDH(d[3],a[3],alpha[3],theta[3],qlim[3]),
    RevoluteDH(d[4],a[4],alpha[4],theta[4],qlim[4]),
    RevoluteDH(d[5],a[5],alpha[5],theta[5],qlim[5])
], name="thorBlue")
T = SE3(0, 0, 0)
robot[0].base = T
T2 = SE3(0, 0, 0)
robot[1].base = T2

#Eliminate delay of first plot----------------------------------------------------
------------------------------
matplotlib.use('Qt5Agg')
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show(block=False)
fig = Figure(figsize=(6, 4), dpi=100)
ax = fig.add_subplot(111, projection='3d')
plot = [0,0]
plt.close()

#Backend thread-------------------------------------------------------------------
---------------
class SecondayThread(threading.Thread):#This thread will be in charge of reading 
and sending data
    def run(self):
            global order
            global normal_mode
            global current
            connect(1,1)
            while app.winfo_exists():
                read()
                if path_mode:
                    try:
                        send_path()
                    except:
                        print('Sending Path failed')
                else:
                    try:
                        send_commands(0)
                        send_commands(1)
                    except:
                        print('Sending Commands failed')
                    time.sleep(0.2)
                time.sleep(0.05)
```

```
#Thread and app declaration-------------------------------------------------------
---------------------------
backend=SecondayThread(daemon=True)
app=tk.Tk()
app.attributes("-fullscreen", True)
app.title("Robotic Station")
app.geometry("1280x720")

#App: tabs-------------------------------------------------------------------------
----------
notebook = ttk.Notebook(app)
tab1 = ttk.Frame(notebook)
tab2 = ttk.Frame(notebook)
tab3 = ttk.Frame(notebook)
tab4 = ttk.Frame(notebook)
notebook.add(tab1, text="Connect")
notebook.add(tab2, text="Controller")
notebook.add(tab3, text="Path planning")
notebook.add(tab4, text="Fullscreen")
style = ttk.Style()
style.configure("TNotebook.Tab", font=("Bold", 12))
style.configure("TNotebook", borderwidth=0)
notebook.pack(fill="both", expand=True)
#App: Main frames------------------------------------------------------------------
-----------------
connection_frame=tk.Frame(tab1)
connection_frame.pack(side="top", fill="both", expand=True)
control_frame=tk.Frame(tab2)
control_frame.pack(side="top", fill="both", expand=True)
path_frame=tk.Frame(tab3)
path_frame.pack(side="top", fill="both", expand=True)
fullscreen_button=tk.Button(tab4,text='Turn off\nFullscreen',font=('Bold',18),
bg=color1, command=lambda: fullscreen_func(), width=25, height=4)
fullscreen_button.place(relx=0.5, rely=0.5, anchor='center')
#App: Connection frame-------------------------------------------------------------
----------------------
connection_button_a=tk.Button(connection_frame, text='Connect
Robot',font=('Bold',18), bg=color1, command = lambda: connect(1,0))
connection_button_a.pack(expand=True,side="left",fill="both", padx=2.5,pady=5)
connection_button_b=tk.Button(connection_frame, text='Connect
Robot',font=('Bold',18), bg=color1, command = lambda: connect(0,1))
connection_button_b.pack(expand=True,side="left",fill="both", padx=2.5,pady=5)

#App: Control frame----------------------------------------------------------------
-------------------
control_label = tk.Label(control_frame, text = "Forward Kinematics Controller",
font=('Bold',14), bg=color2, fg='#FFFFFF')
control_label.grid(row=0, column=1, padx=5, pady=0, columnspan=3, sticky="we")
change_button1 = tk.Button(control_frame, text = "<", font=('Bold',14), bg=color2,
fg='#FFFFFF', command=lambda: change_kinematics())
change_button1.grid(row=0, column=1, padx=5, pady=0, sticky='w')
```

```
change_button2 = tk.Button(control_frame, text = ">", font=('Bold',14), bg=color2,
fg='#FFFFFF', command=lambda: change_kinematics())
change_button2.grid(row=0, column=3, padx=5, pady=0,sticky='e')
acc_label = tk.Label(control_frame, text = "Precision Buttons", font=('Bold',14),
bg=color2, fg='#FFFFFF')
acc_label.grid(row=0, column=0, padx=5, pady=0)
control_frame_acc=tk.Frame(control_frame,bg=color1)
control_frame_acc.grid(row=1, rowspan=2, column=0,  padx=5, pady=5,sticky='nsew')
for i,value in enumerate(list_accuracy):
    precis_button.append(tk.Button(control_frame_acc, text=str(value)+
'º',font=('Bold',20), bd=0, bg=color1, command=lambda t=i: control_accuracy(t)))
    precis_button[i].pack()
precis_button[0].config(bd=3)
stop_button.append(tk.Button(control_frame_acc,
text=stop_labels[0],font=('Bold',14), bg='#FF0000', command=lambda t=0:
control_stop(t)))
stop_button[0].pack(side='bottom', padx=10, pady=10, fill='both')
stop_button.append(tk.Button(control_frame_acc,
text=stop_labels[1],font=('Bold',14), bg='#FF0000', command=lambda t=1:
control_stop(t)))
stop_button[1].pack(side='bottom', padx=10, pady=10, fill='both')

real_time_button = tk.Button(control_frame_acc, text='Real
time:\nOff',font=('Bold',14), bd=0, bg=color1, command=lambda: control_real_time())
real_time_button.pack(side='bottom', padx=10, pady=20, fill='both')

#App: Control frame: robots frames-------------------------------------------------
----------------------------------
gap=tk.Label(control_frame)
gap.grid(row=1, column=2)
for i in range (2):
    control_frame_robots[i]=tk.Frame(control_frame)
    control_frame_robots[i].grid(row=1, column=1+2*i,  padx=5, pady=5, sticky='ns')
    r_lbl_up[i] = tk.Label(control_frame_robots[i], text = robots_labels[i],
bg=color2, fg='#FFFFFF')
    r_lbl_up[i].grid(row=0, column=1, padx=0, pady=0,columnspan=3)
    r_lbl[i] = tk.Label(control_frame, text = "")

    for j, value in enumerate(list_joints):
        joint_label[i].append(tk.Label(control_frame_robots[i], text=value))
        joint_label[i][j].grid(row=j+2, column=0, padx=4)
        joint_slider[i].append(tk.Scale(control_frame_robots[i],
from_=angle_joints_min[j], to=angle_joints_max[j], orient='horizontal',
length=300,width=20, showvalue=True))
        joint_slider[i][j].grid(row=j+2, column=2, padx=5)
        minus_button[i].append(tk.Button(control_frame_robots[i], text='-
',font=('Bold',15), width=2, bg=color1, command= lambda a=i, b=j:
control_minus(a,b)))
        minus_button[i][j].grid(row=j+2, column=1, padx=5)
        plus_button[i].append(tk.Button(control_frame_robots[i],
text='+',font=('Bold',15), width=2, bg=color1, command= lambda a=i, b=j:
control_plus(a,b)))
        plus_button[i][j].grid(row=j+2, column=3, padx=5)

    control_frame_buttons[i]=tk.Frame(control_frame_robots[i])
    control_frame_buttons[i].grid(row=j+3, column=0,  padx=5, pady=5, columnspan=4)
    joint_slider[i][j-1].set(angle_joints_max[j-1])
    joint_slider[i][j].set(angle_joints_max[j])

    undo_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Undo',font=('Bold',15), bg=color1, command = lambda t=i: control_undo(t))
    undo_button[i].grid(row=1, column=0, padx=5, pady=5)
    plot_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Plot',font=('Bold',15), bg=color1, command = lambda t=i: control_plot(t))
    plot_button[i].grid(row=1, column=1, padx=5, pady=5)
    send_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Send',font=('Bold',15), bg=color1, state=tk.DISABLED, command = lambda t=i:
control_send(t))
    send_button[i].grid(row=0, column=0, padx=5, pady=5)
    save_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Save',font=('Bold',15), bg=color1, command = lambda t=i: control_save(t))
    save_button[i].grid(row=0, column=1, padx=5, pady=5)
    execute_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Execute',font=('Bold',15), bg=color1, state=tk.DISABLED, command = lambda
t=i: control_execute(t))
    execute_button[i].grid(row=0, column=2, padx=5, pady=5)
    clear_button[i] = tk.Button(control_frame_buttons[i], width=7,
text='Clear',font=('Bold',15), bg=color1, command = lambda t=i: control_clear(t))
    clear_button[i].grid(row=1, column=2, padx=5, pady=5)

#App: Path frame-----------------------------------------------------------------
----------------
path_label = tk.Label(path_frame, text = "Path Planning", font=('Bold',14),
bg=color2, fg='#FFFFFF')
path_label.grid(row=0, column=0, padx=5, pady=0, sticky="we")
path_file_label = tk.Label(path_frame, text = "File: None", font=('Bold',14),
bg=color2, fg='#FFFFFF')
path_file_label.grid(row=1, column=0, padx=5, pady=0, sticky="w")
path_list=tk.Listbox(path_frame,height=10, width=50, font=('Bold',14))
path_list.grid(row=2, column=0, padx=5, pady=5, sticky="w")
path_list_scrollbar = tk.Scrollbar(path_frame, orient=tk.VERTICAL,
command=path_list.yview, width=40)
path_list_scrollbar.grid(row=2, column=1, sticky=tk.NS)
path_list.config(yscrollcommand=path_list_scrollbar.set)

#App: Path frame: Exec buttons frame---------------------------------------------
------------------------------------
path_frame_exec=tk.Frame(path_frame)
path_frame_exec.grid(row=2, column=2, padx=5, pady=5, sticky="nw",columnspan=30)
```

```
execute_path_button = tk.Button(path_frame_exec, width=13,
text='Execute',font=('Bold',15), bg=color1, command = lambda: path_execute())
execute_path_button.grid(row=0, column=0, padx=5, pady=5, sticky="nw")
stop_path_button = tk.Button(path_frame_exec, width=13,
text='Stop',font=('Bold',15), bg='#FF0000', command = lambda: path_stops())

#App: Path frame: Buttons frame---------------------------------------------------
-------------------------------
path_frame_buttons=tk.Frame(path_frame)
path_frame_buttons.grid(row=3, column=0, padx=5, pady=5, sticky="we",columnspan=30)
open_file_button = tk.Button(path_frame_buttons, width=13, text='Open
File',font=('Bold',15), bg=color1, command = lambda: path_open_file())
open_file_button.grid(row=0, column=0, padx=5, pady=5, sticky="we")
create_file_button = tk.Button(path_frame_buttons, width=13, text='Create
File',font=('Bold',15), bg=color1, command = lambda: path_create_file())
create_file_button.grid(row=0, column=1, padx=5, pady=5, sticky="we")
save_file_button = tk.Button(path_frame_buttons, width=13, text='Save
File',font=('Bold',15), bg=color1, command = lambda: path_save_file())
save_file_button.grid(row=0, column=2, padx=5, pady=5, sticky="we")
rename_file_button = tk.Button(path_frame_buttons, width=13, text='Rename
File',font=('Bold',15), bg=color1, command = lambda: path_rename_file(0))
rename_file_button.grid(row=0, column=3, padx=5, pady=5, sticky="we")
rename_file_entry = tk.Entry(path_frame_buttons, font=('Bold',14))
rename_confirm_button = tk.Button(path_frame_buttons,
text='Accept',font=('Bold',15), bg=color1, command = lambda: path_rename_file(1))
edit_step_button = tk.Button(path_frame_buttons, width=13, text='Edit
Step',font=('Bold',15), bg=color1, command = lambda: path_edit_step())
edit_step_button.grid(row=1, column=0, padx=5, pady=5, sticky="we")
remove_step_button = tk.Button(path_frame_buttons, width=13, text='Remove
Step',font=('Bold',15), bg=color1, command = lambda: path_remove_step())
remove_step_button.grid(row=1, column=1, padx=5, pady=5, sticky="we")
add_step_button = tk.Button(path_frame_buttons, width=13, text='Add Step
Black',font=('Bold',15), bg=color1, command = lambda t=0: path_add_step(t))
add_step_button.grid(row=1, column=2, padx=5, pady=5, sticky="we")
add_step_button_b = tk.Button(path_frame_buttons, width=13, text='Add Step
Blue',font=('Bold',15), bg=color1, command = lambda t=1: path_add_step(t))
add_step_button_b.grid(row=1, column=3, padx=5, pady=5, sticky="we")

#App: Path frame: Steps edit frame------------------------------------------------
----------------------------------
path_frame_edit=tk.Frame(path_frame)
step_order_lbl=tk.Label(path_frame_edit, text = "  N:", font=('Bold',10))
step_order_lbl.grid(row=1, column=0)
step_instruction_lbl=tk.Label(path_frame_edit, text = "  Do:", font=('Bold',10))
step_instruction_lbl.grid(row=1, column=2)
step_robot_lbl=tk.Label(path_frame_edit, text = "  Robot:", font=('Bold',10))
step_robot_lbl.grid(row=1, column=4)
for i in range(len(step_complements_lbl)):
    step_complements_lbl[i]=tk.Label(path_frame_edit, text = "  J"+str(i+1)+":",
font=('Bold',10))
    step_complements_lbl[i].grid(row=1, column=6+i*2)
    step_comp_sel[i]=tk.Entry(path_frame_edit, width=5)
    step_comp_sel[i].grid(row=1, column=7+i*2)
    step_comp_plus[i] = tk.Button(path_frame_edit, text="+", width=5, command=
lambda t=i: path_step_plus(t))
    step_comp_plus[i].grid(row=0, column=7+i*2)
    step_comp_minus[i] = tk.Button(path_frame_edit, text="-", width=5, command=
lambda t=i: path_step_minus(t))
    step_comp_minus[i].grid(row=2, column=7+i*2)
step_complements_lbl[6].config(text = "  Speed:")
step_complements_lbl[7].config(text = "  Accel:")
step_order_sel=tk.Entry(path_frame_edit, width=5)
step_order_sel.grid(row=1, column=1)
step_order_plus = tk.Button(path_frame_edit, text="+", width=5, command= lambda
t=10: path_step_plus(t))
step_order_plus.grid(row=0, column=1)
step_order_minus = tk.Button(path_frame_edit, text="-", width=5, command= lambda
t=10: path_step_minus(t))
step_order_minus.grid(row=2, column=1)
step_instruction=tk.StringVar(path_frame_edit)
step_instruction.set('Move')
step_instructions=['Move','Gripper','Wait']
step_instruction_sel=tk.OptionMenu(path_frame_edit,step_instruction,*step_instructi
ons, command= lambda v: path_instr_sel(v))
step_instruction_sel.grid(row=1, column=3, padx=5)
step_robot=tk.StringVar(path_frame_edit)
step_robot.set('Black')
step_robots=['Black','Blue  ']
step_robot_sel=tk.OptionMenu(path_frame_edit,step_robot,*step_robots)
step_robot_sel.grid(row=1, column=5, padx=5)

#App: Path frame: Steps edit frame 2----------------------------------------------
------------------------------------
path_frame_edit_buttons=tk.Frame(path_frame)
update_step_button = tk.Button(path_frame_edit_buttons, width=13,
text='Update',font=('Bold',15), bg=color1, command = lambda: path_update_step())
update_step_button.grid(row=0, column=0, padx=5, pady=5)
save_step_button = tk.Button(path_frame_edit_buttons, width=13, text='Save
Step',font=('Bold',15), bg=color1, command = lambda: path_save_step())
cancel_step_button = tk.Button(path_frame_edit_buttons, width=13,
text='Cancel',font=('Bold',15), bg=color1, command = lambda: path_cancel_step())
cancel_step_button.grid(row=0, column=4, padx=5, pady=5)
replace_step_button = tk.Button(path_frame_edit_buttons, width=13, text='Replace
Step',font=('Bold',15), bg=color1, command = lambda: path_replace_step())
insert_step_button = tk.Button(path_frame_edit_buttons, width=13, text='Insert
Step',font=('Bold',15), bg=color1, command = lambda: path_insert_step())


#FUNCTIONS SERIAL-----------------------------------------------------------------
-----------------
def read(): #Reads the data sent by both robots, updating ready state if OK is
received
```

```
    global ready
    global robot_a
    global robot_b
    for i in range(2):
        if connected[i]:
            try:
                if serial[i].in_waiting > 0:
                    line = serial[i].readline().decode('utf-8').rstrip()
                    if line == 'OK':
                        ready[i]=1
                        if normal_mode[i]:
                            send_button[i].config(state=tk.NORMAL)
                            execute_button[i].config(state=tk.NORMAL)
                    return line
            except:
                serial[i].close()
                connected[i]=0
                print('Disconnected robot '+robots_labels[i])
        else:
            if robot_a==i:
                connection_button_a.config(state=tk.NORMAL, text= 'Connect Robot')
            elif robot_b==i:
                connection_button_b.config(state=tk.NORMAL, text= 'Connect Robot')
def connect(a,b): #Try to create a serial connection to the robots, and activate
them in the app
    global serial
    global connected
    global orders
    global order
    global seriala
    global serialb
    global robot_a
    global robot_b
    initstring1="M1 2 0 0 0 0 0 4000 4000\n"
    initstring2="M2 2 0 0 0 0 0 4000 4000\n"
    initstring=[initstring1,initstring2]
    ports_detected = ser_tool.comports()
    i=0
    for port, desc, hwid in sorted(ports_detected):
        if len(port)<13:
            if port!='COM7' and port!='COM8' and port!='COM10' and port!='COM11':
                ports[i]=port
                i=i+1
        elif port[11]=='C':
            ports[0]=port
        elif port[11]=='B':
            ports[1]=port
    if a and ports[0]!=-1 and (robot_a==-1 or not connected[robot_a]):
        try:
            seriala=se.Serial(ports[0],38400,timeout=1) #/dev/ttyACM0 vs COM3
            seriala.flush()
            connection_button_a.config(state=tk.DISABLED, text= 'Connecting...')
            linea=""
            while(seriala.isOpen() and linea!="OK"):
                linea = seriala.readline().decode('utf-8').rstrip()
                print(linea)
                if linea=="black":
                    connection_button_a.config(state=tk.DISABLED, text= 'Black 
Robot Connected')
                    n=0

                elif linea=="blue":
                    connection_button_a.config(state=tk.DISABLED, text= 'Blue Robot 
Connected')
                    n=1
            serial[n]=seriala
            robot_a=n
            connected[n]=1
            ready[n]=1
            control_buttons(n,1)
            orders.append(initstring[n])
            order=order+1
            ports[0]=-1
        except:
            print ('Robot not connected in '+ ports[0])

    if b and ports[1]!=-1 and (robot_b==-1 or not connected[robot_b]):
        try:
            serialb=se.Serial(ports[1],38400,timeout=1) #/dev/ttyACM1 vs COM4
            serialb.flush()
            connection_button_b.config(state=tk.DISABLED, text= 'Connecting...')
            lineb=""
            while(serialb.isOpen() and lineb!="OK"):
                lineb = serialb.readline().decode('utf-8').rstrip()
                print(lineb)
                if lineb=="black":
                    connection_button_b.config(state=tk.DISABLED, text= 'Black 
Robot Connected')
                    n=0

                elif lineb=="blue":
                    connection_button_b.config(state=tk.DISABLED, text= 'Blue Robot 
Connected')
                    n=1
            serial[n]=serialb
            robot_b=n
            connected[n]=1
            ready[n]=1
            control_buttons(n,1)
            orders.append(initstring[n])
            order=order+1
            ports[1]=-1
        except:
            print ('Robot not connected in ' + ports[1])
def send_commands(n): #This function sends the commands to the robots whenever it's
possible
    global stopped
    global stop
    global serial
    global current
    global exec_current
    global ready


    if stopped[n]==0:
        if real_time and current==order and connected[n] and ready[n]:
            control_send(n)


        if normal_mode[n] and current<order and orders[current][1]==str(n+1) and
connected[n] and ready[n]:
            serial[n].write(orders[current].encode('utf-8'))
            ready[n]=0
            current=current+1
        if execution_mode[n]:
            print(exec_current[n])
            if exec_current[n]<exec_order[n] and connected[n] and ready[n]:
                serial[n].write(exec_list[n][exec_current[n]].encode('utf-8'))
                ready[n]=0
                exec_current[n]=exec_current[n]+1
            elif exec_current[n]==exec_order[n]:
                execution_mode[n]=0
                current=order
                normal_mode[n]=1
                send_button[n]["state"] = tk.NORMAL
                execute_button[n]["text"] = 'Execute'
    if stop[n]==1 and connected[n]:
        if stopped[n]==0:
            command="STOP\n"
            stopped[n]=1
        else:
            command="CONTINUE\n"
            stopped[n]=0
        serial[n].write(command.encode('utf-8'))
        stop[n]=0
def send_path():#This function is in charge of sending the commands of the path to
each robot, following the path
    global path_i
    global ready
    global wait
    global path_stopped
    global path_stop
    if path_i<path_size  and (connected[0] or connected[1]):
        if not path_stopped:
            line=path_list.get(path_i)
            step=line.split()
            n=int(step[1][1])-1
            instr=step[1][0]
            if ready[n] and (not wait[n] or ready[not n]):
                if wait[n]:
                    wait[n]=0
                if instr=='W':
                    wait[n]=1
                else:
                    line=""
                    for i in range (1,len(step)):
                        line=line+step[i]+" "
                    line=line+"\n"
                    print(line)
                    serial[n].write(line.encode('utf-8'))
                    ready[n]=0
                    path_list.selection_set(path_i)
                    path_list.activate(path_i)
                    path_i=path_i+1
        if path_stop==1:
            if path_stopped==0:
                command="STOP\n"
                path_stopped=1
            else:
                command="CONTINUE\n"
                path_stopped=0
            serial[0].write(command.encode('utf-8'))
            serial[1].write(command.encode('utf-8'))
            path_stop=0
    else:
        path_exec_finish()

#FUNCTIONS CONTROL-----------------------------------------------------------------
------------------
def control_buttons(n,s):#We can activate/desactivate the control robots of each
robot using this function
    if s==0:
        state=tk.DISABLED
    else:
        state=tk.NORMAL
    send_button[n]['state']=state
    execute_button[n]['state']=state

def control_accuracy(n):#Whenever we press a precission button this function update
the accuaricy of the scales buttons
    global accuracy
    if ik==0:
        accuracy=list_accuracy[n]
    else:
        accuracy=list_accuracy_mm[n]
    for i in range(0,len(list_accuracy)):
        precis_button[i].config(bd=0)
    precis_button[n].config(bd=3)

def control_plus(a,b):#Increase slider value in a certain value(accuracy)
    joint_slider[a][b].set(joint_slider[a][b].get()+accuracy)
def control_minus(a,b):#Reduce slider value in a certain value(accuracy)
    joint_slider[a][b].set(joint_slider[a][b].get()-accuracy)

def control_undo(n):#It re-establishes the sliders to the current robot position
    q=[0,0,0,0,0,0]
    if ik==0:
        for i in range (l):
            joint_slider[n][i].set(current_values[n][i])

    if ik==1:
        for i in range (l-3):
            q[i]=current_values[n][i]
        for i in range (l-3,l):
            joint_slider[n][i].set(current_values[n][i])
        pos=angle2pos(n,q)
        for i in range(6):
            pos[i]=round(pos[i],1)
            joint_slider[n][i].set(pos[i])
def control_plot(n):#It plots the robot with the current sliders values
    global plot_mode
    global plot
    if plot_mode==0:
        plot_mode=1
        q=[0,0,0,0,0,0]
        pos=[0,0,0,0,0,0]
        if ik==0:
            for i in range (l-3):
                q[i]=joint_slider[n][i].get()/180*np.pi
        else:
            for i in range (6):
                pos[i]=joint_slider[n][i].get()
            q=pos2angle(n,pos)
            for i in range (l-3):
                q[i]=round(q[i],0)

                q[i]=q[i]/180*np.pi

        robot[n].plot(q, block=False)
        canvas = FigureCanvasTkAgg(plt.gcf(), master=control_frame)
        plot[n]= canvas.get_tk_widget()
        plt.close()
        plot[n].config(width=400, height=400)
        update_pos(n)
        control_frame_robots[not n].grid_forget()
        plot_button[n]['text']='Close'
        plot[n].grid(row=1, column=1+2*(not n),  padx=5, pady=25, sticky='n')
        r_lbl[n].grid(row=1, column=1+2*(not n),  padx=5, pady=75, sticky='s')
    else:
        plot_mode=0
        plot[n].grid_forget()
        r_lbl[n].grid_forget()
        plot_button[n]['text']='Plot'
        control_frame_robots[not n].grid(row=1, column=1+2*(not n),  padx=5,
pady=5, sticky='ns')

def control_send(n):#It add new commands to the strings list that are sent to the
robot
    global order
    global orders
    global current_values
    new_values=[[0,0,0,0,0,0,0,4000,4000],[0,0,0,0,0,0,0,4000,4000]]
    move = False
    gripper = False
    pos=[0,0,0,0,0,0]
    collision=0
    if ik==0:
        for i in range (l):
            new_values[n][i]=joint_slider[n][i].get()
            collision = check_collisions(ik,n,new_values[n])
    else:
        for i in range (6):
            pos[i]=joint_slider[n][i].get()
            collision = check_collisions(ik,n,pos)
        if not collision:
            ang=pos2angle(n,pos)
            q=[0,0,0,0,0,0,0,0,0]
            for i in range (l-3):
                q[i]=int(ang[i]+0.5)
                new_values [n][i]=q[i]
        for i in range (l-3,l):
            new_values[n][i]=joint_slider[n][i].get()
    move,gripper = check_changes(collision,current_values[n],new_values[n])
    send_button[n]["state"] = tk.DISABLED
    execute_button[n]["state"] = tk.DISABLED
    if move==True:
        if n==0:
            movestring="M1"
        else:
            movestring="M2"
        for i in range(l-3):
            movestring=movestring+" "+ str(new_values[n][i])
            current_values[n][i]=new_values[n][i]
        if (current_values[n][7]!=new_values[n][7]):
            movestring=movestring+" S "+ str(new_values[n][7])
            current_values[n][7]=new_values[n][7]
        if (current_values[n][8]!=new_values[n][8]):
            movestring=movestring+" A "+ str(new_values[n][8])
            current_values[n][8]=new_values[n][8]
        movestring=movestring+"\n"
        print(movestring)
        orders.append(movestring)
        order=order+1
        print(order)
        print(movestring)
    if gripper==True:
        current_values[n][6]=new_values[n][6]
        if n==0:
            gripperstring="G1 " + str(new_values[n][6])
        else:
            gripperstring="G2 " + str(new_values[n][6])
        gripperstring=gripperstring+ "\n"
        print(gripperstring)
        orders.append(gripperstring)
        order=order+1
    if move==False and gripper==False:
        send_button[n]["state"] = tk.NORMAL
        execute_button[n]["state"] = tk.NORMAL
def control_save(n):#Save the current sliders value for the robot, for being used
in execution mode or in poth creation
    global exec_order
    global exec_list
    global last_saved
    pos=[0,0,0,0,0,0]
    new_saved = [0,0,0,0,0,0,0,0,0]
    if ik==0:
        for i in range (l):
            new_saved[i]=joint_slider[n][i].get()
            collision = check_collisions(ik,n,new_saved)
    else:
        for i in range (6):
            pos[i]=joint_slider[n][i].get()
            collision = check_collisions(ik,n,new_saved)
        if not collision:
            ang=pos2angle(n,pos)
            q=[0,0,0,0,0,0,0,0,0]
            for i in range (l-3):
                q[i]=int(ang[i]+0.5)
                new_saved[i]=q[i]
            for i in range (l-3,l):
                new_saved[i]=joint_slider[n][i].get()
    move,gripper = check_changes(collision,last_saved[n],new_saved)
    if move or (exec_order[n]==0 and not collision):
        last_saved[n]=new_saved
        movestring="M"
        for i in range (0,l-3):
            movestring=movestring+" "+str(last_saved[n][i])
        movestring=movestring+ " S " + str(last_saved[n][7])+" A "+
str(last_saved[n][8])
        movestring=movestring+"\n"
        exec_list[n].append(movestring)
        print(exec_list[n])
        exec_order[n]=exec_order[n]+1
    elif gripper or exec_order[n]<2:
        last_saved[n][6]=new_saved[6]
        gripperstring="G" + str(last_saved[n][6])
        gripperstring=gripperstring+"\n"
        exec_list[n].append(gripperstring)
        exec_order[n]=exec_order[n]+1

def control_execute(n):#Execute all the saved commands
    global exec_current
    global execution_mode
    global normal_mode
    global current
    if real_time:
        control_real_time()
    if execution_mode[n]==0:
        exec_current[n]=0
        normal_mode[n]=0
        execution_mode[n]=1
        send_button[n]["state"] = tk.DISABLED
        execute_button[n]["text"] = 'FINISH'
    else:
        execution_mode[n]=0
        current=order
        normal_mode[n]=1
        send_button[n]["state"] = tk.NORMAL
        execute_button[n]["text"] = 'Execute'
def control_clear(n):#Clears the execution list of the robot and if we are in
execution mode it also stops it
    global exec_current
    global exec_order
    global current
    global normal_mode
    global exec_list
    empty=[]
    exec_list[n]=empty
    if execution_mode[n]:
        execution_mode[n]=0
        exec_order[n]=0
        exec_current[n]=0
        current=order
        normal_mode[n]=1
        send_button[n]["state"] = tk.NORMAL
        execute_button[n]["text"] = 'Execute'
```

```
def change_kinematics():#It updates sliders values and labels when you change
bewtween Forward and Inverse Kinematics mode
    global ik
    global current_sliders
    global current_sliders_ik
    global total_change
    update=[0,0]
    total_change=1
    new_sliders=[0,0,0,0,0,0]
    if ik==0:
        ik=1
        control_label.configure(text="Inverse Kinematics Controller")
        for i in range(2):
            for j, value in enumerate(list_axis):
                joint_label[i][j].config(text=value)
                joint_slider[i][j].configure(from_=axis_joints_min[j],
to=axis_joints_max[j])
            for j in range(l-3):
                current_sliders[i][j]=joint_slider[i][j].get()
            current_sliders_ik[i]=angle2pos(i,current_sliders[i])
            for j in range(6):
                current_sliders_ik[i][j]=round(current_sliders_ik[i][j],0)
                joint_slider[i][j].set(current_sliders_ik[i][j])
            for j,value in enumerate(list_accuracy_mm):
                precis_button[j].configure(text=str(value)+' mm')

    else:
        ik=0
        control_label.configure(text="Forward Kinematics Controller")
        for i in range(2):

            for j in range(l-3):
                new_sliders[j]=joint_slider[i][j].get()
                if current_sliders_ik[i][j]!=new_sliders[j]:
                    update[i]=1
            for j, value in enumerate(list_joints):
                joint_label[i][j].configure(text=value)
                joint_slider[i][j].configure(from_=angle_joints_min[j],
to=angle_joints_max[j])
            for j,value in enumerate(list_accuracy):
                precis_button[j].configure(text=str(value)+'º')
            if update[i]==1:
                update[i]=0
                current_sliders[i]=pos2angle(i,new_sliders)
            for j in range(6):
                joint_slider[i][j].set(current_sliders[i][j])
    control_accuracy(0)
    total_change=0

def pos2angle(n,pos):#It calculates a possible joints configuration and euler
angles for given end-efffector pose
```

```
    coord=[0,0,0]
    eul=[0,0,0]
    current_q=[0,0,0,0,0,0]
    for i in range (3):
        coord[i]=pos[i]
    for i in range (3,6):
        eul[i-3]=pos[i]
    T=SE3((SE3(coord)*eul2tr(eul, 'deg')))
    for i in range (6):
        current_q[i]=current_values[n][i]
    robot[n].q=current_q
    q=robot[n].ikine_LM(T).q
    for i in range(6):
        q[i]=q[i]*180/np.pi
    return q
def angle2pos(n,q):#It calculates the end-efffector position and euler angles for
given joints values
    pos=[0,0,0,0,0,0]
    ang=[0,0,0,0,0,0]
    for i in range(6):
        ang[i]=q[i]/180*np.pi
    T=robot[n].fkine(ang)
    angles=T.eul()
    for i in range(3):
        pos[i]=T.A[i, 3]
    for i in range(3,6):
        pos[i]=angles[i-3]*180/np.pi
    return pos
def check_collisions(ik,n,q):#It checks if the joint configuration that we want to
send doesn't collide with the floor
    if ik==0:
        pos=angle2pos(n,q)
    else:
        pos=q
    if pos[2]<0:
        r_lbl_up[n].config(text=robots_labels[n]+"\nERROR: Robot collision",
bg='#FF0000')
        return 1
    else:
        r_lbl_up[n].config(text=robots_labels[n], bg=color2)
        return 0
def check_changes(collision,last_values, new_values):#It checks if the joints
configuration and/or the gripper value have changed
    move=False
    gripper=False
    if not collision:
        for i in range(l-3):
            if (last_values[i]!=new_values[i]):
                move=True

    if (last_values[6]!=new_values[6]):
        gripper=True
    return move,gripper

def update_pos(n): #Update text with the position of the robot in real time
    q=[0,0,0,0,0,0]
    real_time_ang=q
    if ik==0:
        for i in range (6):
            q[i]=joint_slider[n][i].get()
        real_time_pos=angle2pos(n, q)
        for i in range (6):
            real_time_pos[i] = str(round(real_time_pos[i], 1))
        textt="x="+real_time_pos[0]+"    y="+real_time_pos[1]+"    z="+real_time_po
s[2]+'\n'
        textt=textt+"Rx="+real_time_pos[3]+"    Ry="+real_time_pos[4]+"    Rz="+rea
l_time_pos[5]
        r_lbl[n].configure(text=textt)
    else:
        for i in range (6):
            q[i]=joint_slider[n][i].get()
        ang=pos2angle(n,q)
        for i in range (6):
            real_time_ang[i] = int(ang[i]+0.5)
        textt="J1="+str(real_time_ang[0])+"    J2="+str(real_time_ang[1])+"    J3="
+str(real_time_ang[2])+'\n'
        textt=textt+"J4="+str(real_time_ang[3])+"    J5="+str(real_time_ang[4])+"
 J6="+str(real_time_ang[5])
        r_lbl[n].configure(text=textt)
    '''def scale_callback(v,n):
        if total_change==0:
            update_pos(n)
    '''

def control_stop(n):#It stops/continues robot movement in any moment you call it
    global stop
    if stopped[n]==0 and connected[n]:
        stop[n]=1
        stop_button[n].config(bg=color2,text=stop_labels[n+2])
    if stopped[n]==1 and connected[n]:
        stop[n]=0
        stop_button[n].config(bg='#FF0000',text=stop_labels[n])
def control_real_time():#It activates/deactivates real time mode
    global real_time
    global normal_mode
    global execution_mode
    if real_time==0:
        real_time=1
        normal_mode[0]=1
        normal_mode[1]=1
        execution_mode[0]=0
        execution_mode[1]=0
        real_time_button.config(bd=3,text="Real time:\nOn")
    else:
        real_time=0
        real_time_button.config(bd=0,text="Real time:\nOff")

#FUNCTIONS PATH------------------------------------------------------------------
---------------
def path_open_file():#It tries to open a text file for loading a path in the app
    global text_file_dir
    try:
        text_file_dir = filedialog.askopenfilename(initialdir='paths/',title='Open
Path (text file)', filetypes=[('Text Files', '*.txt')])
        start_index = text_file_dir.find("paths")
        if start_index != -1:
            result_string = text_file_dir[start_index+6:]
        path_file_label.config(text="File: "+result_string)
        text_file = open(text_file_dir, 'r')
        path_text = text_file.read()
        text_file.close()
        current_path=path_text.splitlines()
        path_list.delete(0, 'end')
        for i, step in enumerate(current_path):
            path_list.insert(i,str(i+1)+" "+step)
    except:
        print('File not open')
def path_create_file():#It creates a new file with our current path, using as name
'Path'+current date+current time
    global text_file_dir

    now = datetime.datetime.now()
    date_time = now.strftime("Path %Y-%m-%d_%H.%M.%S")
    directory = "paths/"

    text_file_dir = directory+ f"{date_time}.txt"
    path_file_label.config(text="File: "+text_file_dir[6:])
    path_save_file()
def path_save_file():#You can save the current path in the last file that you
openned/created, if not it creates a new file
    if text_file_dir!= "":
        with open(text_file_dir, "w") as file:

            for item in path_list.get(0, tk.END):

                words = item.split()

                new_line = " ".join(words[1:])
                file.write(new_line + "\n")
    else:
        path_create_file()

def path_rename_file(x):#Rename the file you are working on
    global text_file_dir
```

```
        if x==0:
            if not rename_file_entry.grid_info():
                rename_file_entry.grid(row=0, column=4, padx=5, pady=5)
                rename_confirm_button.grid(row=0, column=5, padx=5, pady=5)
                rename_file_entry.delete(0, tk.END)
            else:
                rename_file_entry.grid_forget()
                rename_confirm_button.grid_forget()
        else:
            new_name = rename_file_entry.get()
            start_index = text_file_dir.find("paths")
            if start_index != -1:
                old_name = text_file_dir[start_index+6:]
            if new_name!="" and new_name!=old_name:
                new_dir="paths/" + new_name + ".txt"
                os.rename(text_file_dir, new_dir)
                text_file_dir=new_dir
                path_file_label.config(text="File: "+text_file_dir[6:])
                rename_file_entry.grid_forget()
                rename_confirm_button.grid_forget()

    def path_edit_step():#It opens an editting menu to edit the step you've clicked on
        global step_robot_option
        global edit_item
        for edit_item in path_list.curselection():
            path_frame_edit.grid(row=4, column=0, padx=5, pady=5,
    sticky='w',columnspan=30)
            path_frame_edit_buttons.grid(row=5, column=0, padx=5, pady=5,
    sticky='w',columnspan=30)
            if not save_step_button.grid_info():
                save_step_button.grid(row=0, column=1, padx=5, pady=5)
                replace_step_button.grid_forget()
                insert_step_button.grid_forget()
            step=path_list.get(edit_item).split()
            step_order_sel.delete(0, tk.END)
            step_order_sel.insert(0, step[0])
            if step[1][1]=='1':
                step_robot.set('Black')
            elif step[1][1]=='2':
                step_robot.set('Blue  ')
            if step[1][0]=='M':
                step_instruction.set('Move')
                path_edit_regrid(8,0)
            elif step[1][0]=='G':
                step_instruction.set('Gripper')
                path_edit_regrid(1,8)
            elif step[1][0]=='W':
                step_instruction.set('Wait')
                path_edit_regrid(0,8)
            values = step[2 : ]
            update_step(values)
```

```
def path_remove_step():#It removes the step you've clicked on from the path
(reordering the rest of steps)
    for item in path_list.curselection():
        path_list.delete(item)
        reorder_steps(item,0)
    finish_edit_step()
def path_add_step(n):#It opens the editting menu, that allows you to add new steps
    step_order_sel.delete(0, tk.END)
    step_order_sel.insert(0, path_list.size()+1)
    if not replace_step_button.grid_info():
        save_step_button.grid_forget()
        replace_step_button.grid(row=0, column=2, padx=5, pady=5)
        insert_step_button.grid(row=0, column=3, padx=5, pady=5)
    path_frame_edit.grid(row=4, column=0, padx=5, pady=5, sticky='w',columnspan=30)
    path_frame_edit_buttons.grid(row=5, column=0, padx=5, pady=5,
sticky='w',columnspan=30)
    if n==0:
       step_robot.set('Black')
    else:
         step_robot.set('Blue  ')
    update_step(last_saved[n])

#FUNCTIONS PATH: Edit functions
def finish_edit_step():#This function re-configure the gui when you finish the
edition of a step
    path_frame_edit.grid_forget()
    path_frame_edit_buttons.grid_forget()
def reorder_steps(item,x):#This function serves to reorder the paths steps. It is
used in other functions: remove, replace, insert,...
    for i in range(item,path_list.size()):
        step=path_list.get(i).split()
        if x==0:
            step[0]=str(int(step[0])-1)
        else:
            step[0]=str(int(step[0])+1)
        line=""
        for j in range(len(step)):
            line=line+step[j]+" "
        path_list.delete(i)
        path_list.insert(i,line)
def update_step(val):#It is used to update the step values in the editting menu to
the last saved in the controller tab
    current_inst=step_instruction.get()
    if current_inst=='Move':
        step_instruction.set('Move')
        step_complements_lbl[0].config(text = "  J1: ")
        for i in range(8):
            step_comp_sel[i].delete(0, tk.END)
            if i<6:
                step_comp_sel[i].insert(0, val[i])
            else:
                step_comp_sel[i].insert(0, val[i+1])
        if len(val)==10:
            step_comp_sel[i].delete(0, tk.END)
            step_comp_sel[i].insert(0, val[i+2])
        path_edit_regrid(8,0)

    elif current_inst=='Gripper':
        step_instruction.set('Gripper')
        step_complements_lbl[0].config(text = "  Degrees:")
        step_comp_sel[0].delete(0, tk.END)
        if len(val)>6:
            step_comp_sel[0].insert(0, val[6])
        else:
            step_comp_sel[0].insert(0, val[0])
        path_edit_regrid(1,8)

    elif current_inst=='Wait':
        step_instruction.set('Wait')
        path_edit_regrid(0,8)

#FUNCTIONS PATH: Edditing steps functions
def path_instr_sel(v):#It just calls path_update_step() when the instructions menu
of the editting manu is used
    path_update_step()
def path_step_plus(n):#It increase current value in 5 units
    if n==10:
        try:
            number= int(step_order_sel.get())
        except:
            number=0
        number=number+1
        step_order_sel.delete(0, tk.END)
        step_order_sel.insert(0, str(number))
    else:
        try:
            number= int(step_comp_sel[n].get())
        except:
            number=0
        if n>5:
            number=number+1000
        else:
            number=number+5
        step_comp_sel[n].delete(0, tk.END)
        step_comp_sel[n].insert(0, str(number))
def path_step_minus(n):#It decrease current value in 5 units
    if n==10:
        number= int(step_order_sel.get())
        number=number-1
        step_order_sel.delete(0, tk.END)
        step_order_sel.insert(0, str(number))
    else:
        number= int(step_comp_sel[n].get())
        if n>5:
            number=number-1000
        else:
            number=number-5
        step_comp_sel[n].delete(0, tk.END)
        step_comp_sel[n].insert(0, str(number))

def insert(x):#It inserts a new step in the path. It's called from
path_insert_step() and path_replace_step()
    collision = 0
    step=[]
    step.append(step_order_sel.get())
    inst=step_instruction.get()[0]
    pos = [0,0,0,0,0,0]
    if step_robot.get()=='Black':
        n='1'
    else:
        n='2'
    step.append(inst+n)
    if inst=='M':
        for i in range(6):
            step.append(step_comp_sel[i].get())
            pos[i]=int(step[i+2])
        collision = check_collisions(0,int(n)-1,pos)
        if not collision:
            step.append('S')
            step.append(step_comp_sel[6].get())
            step.append('A')
            step.append(step_comp_sel[7].get())
            line=""
            for i in range(12):
                line=line+step[i]+" "
    elif inst=='G':
        step.append(step_comp_sel[0].get())
        line=step[0]+" "+step[1]+" "+step[2]
    elif inst=='W':
        line=step[0]+" "+step[1]
    if not collision:
        path_label.config(text = "Path Planning", bg=color2)
        new_item=int(step[0])-1
        path_list.insert(new_item+x,line)
        reorder_steps(new_item+1,1)
    else:
        path_label.config(text = "Path Planning: ERROR: Robot Collision",
bg='#FF0000')
    return not collision
def path_edit_regrid(n,m):#It changes our editting menu, depending of the command
we have to edit
    for i in range(n):
        if not step_complements_lbl[i].grid_info():
            step_complements_lbl[i].grid(row=1, column=6+i*2)
            step_comp_sel[i].grid(row=1, column=7+i*2)
            step_comp_plus[i].grid(row=0, column=7+i*2)
            step_comp_minus[i].grid(row=2, column=7+i*2)
    for i in range(n,m):
        if step_complements_lbl[i].grid_info():
            step_complements_lbl[i].grid_forget()
            step_comp_sel[i].grid_forget()
            step_comp_plus[i].grid_forget()
            step_comp_minus[i].grid_forget()
def path_exec_finish():#It closes the edditting menu
    global wait
    global path_mode
    open_file_button['state']=tk.NORMAL
    create_file_button['state']=tk.NORMAL
    save_file_button['state']=tk.NORMAL
    rename_file_button['state']=tk.NORMAL
    edit_step_button['state']=tk.NORMAL
    remove_step_button['state']=tk.NORMAL
    add_step_button['state']=tk.NORMAL
    add_step_button_b['state']=tk.NORMAL
    execute_path_button.config(text='Execute', bg=color1, fg='#000000')
    stop_path_button.grid_forget()
    path_mode=0
    path_list.selection_clear(0, tk.END)

#FUNCTIONS PATH: Edditing frame buttons functions
def path_cancel_step():#It closes the editting menu
    finish_edit_step()
def path_update_step():#It updates the edditing menu variables to the last values
saved
    if step_robot.get()=="Black":
        n=0
    else:
        n=1
    update_step(last_saved[n])
def path_save_step():#It saves the step you were editting, with possibility of
changing the order of the step
    if edit_item <= int(step_order_sel.get()):
        if insert(1):
            path_list.delete(edit_item)
            reorder_steps(edit_item,0)
            finish_edit_step()
    else:
        if insert(0):
            path_list.delete(edit_item+1)
            reorder_steps(edit_item+1,0)
            finish_edit_step()

def path_insert_step():#It inserts a new step wherever you want, reordering the
rest of steps
```

```
    if insert(0):
        finish_edit_step()
def path_replace_step():#It replaces an old step to a new one
    if insert(1):
        item=int(step_order_sel.get())-1
        if item+1<path_list.size():
            path_list.delete(item)
            reorder_steps(item,0)
        finish_edit_step()

#FUNCTIONS PATH EXEC---------------------------------------------------------------
--------------------
def path_execute():#It activates the path execution mode, controlling both robots
    global path_i
    global path_size
    global path_mode
    global wait
    global path_stopped
    global path_stop
    if path_list.size()>0:
        if not stop_path_button.grid_info():
            wait=[0,0]
            path_i=0
            path_stopped=0
            path_stop=0
            path_list.selection_clear(0, tk.END)
            path_size=path_list.size()
            path_mode=1
            finish_edit_step()
            open_file_button['state']=tk.DISABLED
            create_file_button['state']=tk.DISABLED
            save_file_button['state']=tk.DISABLED
            rename_file_button['state']=tk.DISABLED
            edit_step_button['state']=tk.DISABLED
            remove_step_button['state']=tk.DISABLED
            add_step_button['state']=tk.DISABLED
            add_step_button_b['state']=tk.DISABLED
            execute_path_button.config(text='Finish', bg=color2, fg='#FFFFFF')
            stop_path_button.config(text='Stop', bg='#FF0000', fg='#000000')
            stop_path_button.grid(row=1, column=0, padx=5, pady=5, sticky="nw")
        else:
            path_exec_finish()
def path_stops():#It stops/continues the path_execution
    global path_stop
    path_stop=1
    if stop_path_button['text']=='Stop':
        stop_path_button.config(text='Continue', bg=color2, fg='#FFFFFF')
    else:
        stop_path_button.config(text='Stop', bg='#FF0000', fg='#000000')
```

```
#FUNCTIONS GENERAL USE--------------------------------------------------------------
----------------------
def fullscreen_func():#Enables/Disables fullscreen
    if app.attributes("-fullscreen")==True:
        fullscreen_button['text']='Turn on\nFullscreen'
        app.attributes("-fullscreen", False)
    else:
        fullscreen_button['text']='Turn off\nFullscreen'
        app.attributes("-fullscreen", True)

#Initialize app---------------------------------------------------------------------
---------------
backend.start()
app.mainloop()
```

Appendix 2. Robot controller code: Main_controller.ino

```
#include <math.h>
#include <AccelStepper.h>
#include <MultiStepper.h>
#include "Config_and_variables.h"
#include "Nano_communication.h"
#include "Serial_communication.h"
#include "Movement.h"

void setup()
{
  pinMode(EN_PIN, OUTPUT);
  digitalWrite(EN_PIN, LOW);
  for(int i=0; i<7; i++)
  {
    pinMode(STEP_PIN[i], OUTPUT);
    pinMode(DIR_PIN[i], OUTPUT);
    digitalWrite(STEP_PIN[i], LOW);
    digitalWrite(DIR_PIN[i], LOW);
    motor[i] = AccelStepper(AccelStepper::DRIVER, STEP_PIN[i], DIR_PIN[i]);
    motor[i].setMaxSpeed(speed);
    motor[i].setAcceleration(accel);
    motors.addStepper(motor[i]);
  }
  delay(3000);
  Serial.begin(38400);
  delay(500);
  Serial3.begin(38400);
  feedback();
  newTarget=1;
  Serial.print("blue\n");
  MoveToPosition();
}
void loop()
{

  read();
  if (!stopped)
  {
    feedback();
    MoveToPosition();
    MoveGripper();
  }
}
```

Appendix 3. Robot controller code: Config_and_variables.h

```
#define EN_PIN 40
int STEP_PIN[] = {28,26,22,23,25,27,24}; //2motor de art2 el ultimo
int DIR_PIN[] = {36,34,30,31,33,35,32}; //2motor de art2 el ultimo

bool moveGripper=0;
bool stopped=0;
bool moving=0;
bool running=0;
bool runningArt6=0;
bool newTarget=0;
float AnglesNext[7]={0,0,0,0,0,0,0};
int gripper=0;
String inString="";
int AnglesToSteps[7]={3200/180*2,284.44,3200/180*16,3200/180,3200/180,3200/180,284.44};
AccelStepper motor[7];
MultiStepper motors;
int speed = 4000; //MAX 4000 CON 3200 DE STEPSAngles
int accel = 1000;
int delayy = 1;
long time=0;
long steps[7]={0,0,0,0,0,0,0};
long AngleArt5;
long AngleArt6;
```

Appendix 4. Robot controller code: Movement.h

```
void MoveGripper()
{
  if (moveGripper)
  {
    moveGripper=0;
    Serial3.print("G");
    Serial3.print(gripper);
    Serial3.print("\n");
    delay(1000);
    Serial.print("OK");
  }
}
```

```
void MoveToPosition(){

  while (motors.run()) {
    while (motors.run()){
      if (Serial.available())
      {
        read();
        while(stopped) read();
      }
    }

    feedback();

  }
  if (running && !newTarget)
  {
    running=0;
    Serial.print("OK");
    Serial.print("\n");
  }
  else if (runningArt6 && !newTarget)
  {
    runningArt6=0;
    running=1;

    int i;
    for(i=0; i<7; i++) {
      if (i!=4 && i!=5)
      {
        steps[i]=AnglesNext[i]*AnglesToSteps[i];
      }
      motor[i].setMaxSpeed(speed);
      motor[i].setAcceleration(accel);
    }
    AnglesArt56();
    steps[4]+=AnglesNext[4]*AnglesToSteps[4]-AngleArt5;
    steps[5]+=AngleArt5-AnglesNext[4]*AnglesToSteps[4];
    motors.moveTo(steps);
  }
  else if (newTarget)
  {
    newTarget=0;
    running=0;
    runningArt6=1;

    int i;
    for(i=0; i<7; i++) {
      if (i!=4 && i!=5)
      {
        steps[i]=motor[i].currentPosition();
      }
      motor[i].setMaxSpeed(speed);
      motor[i].setAcceleration(accel);
    }
    AnglesArt56();
    steps[4]+=AnglesNext[5]*AnglesToSteps[5]-AngleArt6;
    steps[5]+=AnglesNext[5]*AnglesToSteps[5]-AngleArt6;

    motors.moveTo(steps);
  }
}
```

Appendix 5. Robot controller code: Nano_communication.h

```
void feedback()
{
  bool received=0;
  const int size = 100;
  int i;
  char inChar1='\0';
  char inArray1 [size];
  String inString1="";
  String token1 [7];
  //Serial3.print('P');

  delay(delayy);

  while (Serial3.available() > 0){
      inChar1=(char)Serial3.read();
    if (inChar1 != '\n')
    {
      inString1 += inChar1;
    }
    else
    {
      float current;
      float newPos;
      inString1.toCharArray(inArray1, size);
      token1[i] = strtok(inArray1, " ");
      while (token1[i] != NULL) {
        i++;
        token1[i] = strtok(NULL, " ");
      }
      for(int i=0;i<5;i++)
      {
        if (i<4 || (abs(AnglesNext[4])+abs(AnglesNext[5]))<165)
        {
          current = motor[i].currentPosition();
          newPos = token1[i].toFloat()*AnglesToSteps[i];
          if (abs(current-newPos)<2*AnglesToSteps[i])
            motor[i].setCurrentPosition(newPos);
            if (i==1) motor[6].setCurrentPosition(newPos);
        }
      }
      inString="";
    }
  }
}
```

Appendix 6. Robot controller code: Serial_communication.h

```
String output(float v[], String v2)
{
   String output = v2+": x="+String(v[0])+", y="+String(v[1])+", z="+String(v[2])+", X="+String(v[3])+", Y="+String(v[4])+", Z="+String(v[5]);
  return(output);
}
void AnglesArt56()
{
  AngleArt6=(motor[4].currentPosition()+motor[5].currentPosition())/2;
  AngleArt5=motor[4].currentPosition()-AngleArt6;
}
void read()
{
  if (Serial.available() > 0)
  {
    int i=0;
    const int size = 100;
    char inArray [size];
    char inChar;
    String token [15];
    while (Serial.available() > 0){
      inChar=(char)Serial.read();
      if (inChar != '\n')
      {
        inString += inChar;
      }
      else
      {
        inString.toCharArray(inArray, size);
        token[i] = strtok(inArray, " ");
        while (token[i] != NULL) {
          i++;
          token[i] = strtok(NULL, " ");
        }
        switch(token[0].charAt(0)) {
          case 'M': //MoveRobot
            for(int i=0;i<6;i++)
            {
              AnglesNext[i]=token[i+1].toFloat();
            }
            AnglesNext[6]=AnglesNext[1];
            if (token[7] == "S")
              speed=token[8].toInt();
            if (token[7] == "A")
              accel=token[8].toInt();
            if (token[9]=="A")
              accel=token[10].toInt(); //Move Angle1 Angle2 Angle3 Angle4 Angle5 Angle6 S speed A
accel
            Serial.print("received");
            Serial.print("\n");
            newTarget=1;
          break;

          case 'G': //GRIPPER
            gripper=token[1].toInt();
            moveGripper=1;
            Serial.print(gripper);
            Serial.print("\n");
          break;
          case 'S': //STOP
            stopped=1;
            Serial.print("OK\n");
          break;
          case 'C': //CONTINUE
            stopped=0;
            Serial.print("OK\n");
          break;
          case 'P': //Position?
            feedback();
            String output = "Position";
            for (int j=0;j<4;j++)
            {
              output+=" ";
              output += motor[j].currentPosition();
            }
            AnglesArt56();
            output+=" ";
            output+=AngleArt5;
            output+=" ";
            output+=AngleArt6;
            Serial.print(output);
            Serial.print("\n");
          break;
          case 'D': //Desactivate Motors
            stopped=1;
            digitalWrite(EN_PIN, HIGH);
            Serial.print("OK");
            Serial.print("\n");
          break;
          case 'A': //Activate Motors
```

```
          stopped=0;
          digitalWrite(EN_PIN, LOW);
          Serial.print("OK");
          Serial.print("\n");
         break;
        }
        inString="";
      }
    }
  }

}
```

Appendix 7. Encoders controller code.

```
#include <Servo.h>
#define GRIPPER 3

int i=0;
int servo=0;
int start=0;

int SENSOR[5]= {A2, A3, A4, A5, A6};
float angle[5];

String inString;
Servo servoMotor;

void setup()
{
  servoMotor.attach(GRIPPER);
  Serial.begin(38400);
  for (i=0; i<5; i++)
  {

    pinMode(SENSOR[i], INPUT);

  }
}
void loop()
{

  read();

}
```

```
void read()
{
  char inChar;
  if (Serial.available() > 0)
  {
    inChar=(char)Serial.read();
    switch (inChar)
    {
      case 'P':
      {
        for (i=0; i<6; i++)
        {
          angle[i] = analogRead(SENSOR[i])/1023*333.33-166.67;
          Serial.print(angle[i]);
          Serial.print(" ");
        }
        Serial.print("\n");
      }
      break;
      case 'G':
      {
        inString="";
        delay(1);
        while(inChar!= '\n' && Serial.available())
        {
          inChar=Serial.read();
          if (inChar!= '\n')
            inString += inChar;
        }

        servo=140-140*inString.toInt()/100;
        Serial.print(servo);
        servoMotor.write(servo);
      }
      break;
    }
  }
}
```