Bachelor's Thesis

Bachelor's Degree in Industrial Technology Engineering

Control of a robotic arm using an Omega2+ module

REPORT

Author: Daniel Vázquez Navarro
Director: Manuel Moreno Eguilaz
Term: 2017/2018 – Autumn semester

Barcelona School of Industrial Engineering
Review

The aim of this project has been to introduce the Omega2 [1] module in the Department of Electronic Engineering (EEL) at ETSEIB. This project can beneficial for further students who want to use the module as this report can be used as an Omega user guide.

Moreover, the module has been introduced in a real project by replacing the current control module by the Omega board. The module has been successfully adapted to a current robot. Some adaptation has been needed like developing a library to fit perfectly the module with the other components.

Furthermore, a web-based API has been developed to allow the user to control the robot. By using it, the user can control the robot in an easy and intuitive way. This interface allows to control each servomotor in independently and to make horizontal and vertical movements, which are the combination of two servomotors. In addition, the user can press a button that will run a pre-loaded automatic movement that will show the robot arm potential.

The result is a fully functional low-scale robot that can be controlled by everybody. Due to this tiny size, it can be conceived as toy but it could be easily, in a bigger scale, an industrial robot that could perfectly be in a factory helping people to achieve their work.
Summary

REVIEW .................................................................................................................. 2
SUMMARY ............................................................................................................... 3

1. GLOSSARY ........................................................................................................ 5

2. PREFACE .......................................................................................................... 7
  2.1. Origin of the project ....................................................................................... 7
  2.2. Motivation ...................................................................................................... 7
  2.3. Previous requirements ................................................................................. 8

3. INTRODUCTION ................................................................................................. 9
  3.1. Objectives of the project ............................................................................. 9
  3.2. Scope of the Project .................................................................................... 9

4. OMEGA2 ............................................................................................................ 11
  4.1. Introduction .................................................................................................. 11
  4.2. Comparison with other modules ................................................................. 13
  4.3. Omega2 Docks ............................................................................................ 15
  4.4. Setup ............................................................................................................ 17
  4.5. Connecting to the Omega .......................................................................... 18
     4.5.1. The Command-Line Interface ............................................................... 18
     4.5.2. Connecting with SSH ........................................................................... 18
     4.5.3. Connecting with Omega’s Console ....................................................... 20
  4.6. Using/developing with the Omega ............................................................. 21

5. ROBOTIC ARM .................................................................................................. 25
  5.1. Servomotor & PWM signal ......................................................................... 26
  5.2. I2C ............................................................................................................... 28
     5.2.1. Onion I2C Python Module ................................................................. 29
  5.3. PCA9685 .................................................................................................... 31
     5.3.1. PCA Library ....................................................................................... 33
  5.4. Robot mechanism ....................................................................................... 36
     5.4.1. Main mechanism ............................................................................... 36
     5.4.2. Hand mechanism ............................................................................. 39

6. SOFTWARE ....................................................................................................... 42
  6.1. Description of the global architecture ....................................................... 42
  6.2. How to run .................................................................................................. 43
6.3. Main program ................................................................. 45
  6.3.1. robot.py ................................................................. 47
  6.3.2. baseservos.py ....................................................... 51
  6.3.3. servos.py ............................................................. 52
  6.3.4. mecanisme.py ....................................................... 53
  6.3.5. interpolacions.py .................................................. 55
6.4. Web-Based API ............................................................ 56
  6.4.1. API interface .......................................................... 56
  6.4.2. API software .......................................................... 58

7. PROJECT SCHEDULE .......................................................... 62

8. BUDGET ..................................................................... 64

9. ENVIRONMENTAL IMPACT .................................................. 66

CONCLUSIONS ..................................................................... 67

ACKNOWLEDGEMENTS .......................................................... 68

REFERENCES .......................................................................... 69
1. Glossary

**API (Application Programming Interface):** Set of subroutine definitions, protocols, and tools for building application software.

**App:** Smartphone application.

**CPU (Central Processing Unit):** Electronic circuitry in a computer that carries out the instructions of a computer program.

**CSS (Cascading Style Sheets):** Style sheet language used for describing the presentation of a document written in a markup language.

**ETSEIB:** Escola Tècnica Superior d'Enginyeria Industrial de Barcelona.

**GPIO (General-purpose input/output):** Generic pin on an integrated circuit or computer board.

**HDMI (High-Definition Multimedia Interface):** Audio and video interface.

**HTML (Hypertext Markup Language):** Standard markup language for creating web pages and web applications.

**I2C (Inter-Integrated Circuit):** Multi-master and multi-slave serial computer bus.

**I2S (Inter-IC Sound):** Electrical serial bus interface standard used for connecting digital audio devices together.

**IoT (Internet of Things):** Network of physical devices which are connected and exchange data.

**IP (Internet Protocol) address:** Numerical label assigned to each device connected to a computer network.

**jQuery:** Cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

**JSON (JavaScript Object Notation):** Open-standard file format that uses human-readable text.

**LED (Light-Emitting Diode):** Two-lead semiconductor light source.

**MAC (Media Access Control) address:** Unique identifier assigned to network interfaces.
SD (Secure Digital): Non-volatile memory card format.

MOQ: Minimum Order Quantity

OOP (Object-oriented programming): Programming based on the concept of "objects".

OS (Operating System): Computer system software that manages the hardware and software of a computer.

PLA (Polylactic acid): Biodegradable and bioactive thermoplastic.

PWM (Pulse-width modulation): Modulation technique used to encode a message into a pulsing signal.

RAM (Random-access memory): Form of computer data storage that stores data and machine code currently being used.

SoC (System on Chip): Integrated circuit that integrates all components of a computer or other electronic systems.

SPI (Serial Peripheral Interface): Synchronous serial communication interface specification used for short distance communication.

SSH (Secure Shell): Network protocol for operating network services securely over an unsecured network.

UART (Universal Asynchronous Receiver-Transmitter): Computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.

USB (Universal Serial Bus): Industry standard that defines for connection, communication, and power supply between computers and devices.
2. Preface

2.1. Origin of the project

Professor Manuel Moreno Eguílaz gave me the opportunity and encouraged me to perform the End of Bachelor Degree Project focused on the Omega2 module [1]. The preliminary investigations on the Omega 2+ reapproved me the idea to develop a project based on electronics and one micro-robotic arm [2] available in the Department of Electronic Engineering.

This project has been a good opportunity to know about the module, its application and its potential since it is has been a new acquisition from the Department of Electronic Engineering. Furthermore I strongly believe that the Omega2 will probably be used in some courses or other projects in the near future.

2.2. Motivation

The main motivation has been to apply my knowledge acquired during the Degree. Proving that what I have learned can take you forward into a full journey that ends with a tangible and visual final product.

This challenge captivated me because I am interested in the robotics. I started my adventure in the robotic field when I was in High School doing my Final Project based on the Lego Mindstorms [3]. During this degree, I had also the opportunity to make a delta robot. So, this project has been a good opportunity to keep growing my interest into robotics.

Furthermore, this world is getting every day more and more into the “Internet of Things” (IoT). We want it or not, the future will tend towards human interfacing and controlling every single detail from their smartphones or tablets. That is why nowadays, an engineer must know about it and should take a part into it. This project has been an excellent opportunity to make some first steps on the IoT environment.
2.3. Previous requirements

To carry out this project, a global knowledge on different areas of Engineering has been required. I have had the opportunity to use some knowledge acquired during the Degree such as:

- Python programming language and OOP from *Fonaments d'Informàtica* and *Informàtica*.
- Structure analysis and design from *Teoria de Màquines i Mecanismes*.
- Basic electronics from *Electrònica*.

Besides the knowledge acquired during the degree, I have also been required some knowledge on web designing like HTML, JavaScript or CSS.
3. Introduction

3.1. Objectives of the project

Amongst all the objectives, this project is principally focused on finding out what the Omega2 [1] module is. What I have tried was to build a fully project based on this module.

The main target was how this module worked out and get to know it.. So, I have started introducing the module and its benefits. But I did not only want to show what it is, but I also wanted to compare it with some other products available on the market.

I did not stop once I knew what it was, I also wanted to step on how it van be used. During this report I have explained how to configure the module and how to access it. This project also aims to be a guidebook for whoever wants to start working with an Omega2 module.

It is important to know the theory, but we must apply it. That is why another objective has been the replacement of the current controlling module, in an actual project, by the Omega2. This project was a micro-robotic arm that was built and developed by previous ETSEIB’s students. The objective was not to make any improvements on the robot, only change the controlling module, but I have been forced to make some changes as the adaptation of a driver (5.3.1 PCA Library) or the replacement of a structural part. In order to help whoever who desires to continue using the robot, I have also used this project to create a compilation of all the previous information about the robot descripted in the previous projects [2], [4] and [5].

Finally, since the current robot controlling system was not compatible with Omega2, I have implement a new web-based API.

3.2. Scope of the Project

The scope of this project has been:

- Get familiar with Omega2.
- Understand why we are using this module.
- Write a user guide for forward project/purpose with Omega2 module.
- Annalise the current project status.
- Make a compilation of current information about the robot.
- Reduce the price of the robot (electronic components).
- Get familiar with electronic communication and field (I2C, PWM, etc…).
- Adapt the code to the current requirements.
- Develop an API to control the robot in a manual and automatic way.
4. Omega2

4.1. Introduction

In order to understand why I am building a fully project based on the Omega2 we firstly need to comprehend what this module is.

The Omega2 [1] is a computer running Linux. We can think of the Omega2 as a tiny Linux server with Wi-Fi so it can be defined as an IoT computer. An important benefit of running Linux is that the Omega2 can be programmed with whatever language we want (Python, Node, C++, php, etc...)

It is a Linux computer designed specifically for building connected hardware applications. “It combines the tiny form factor (1/4 the size of the Raspberry Pi, and less than 1/3 the size of the Arduino Uno) and power-efficiency of the Arduino, with the power and flexibilities of the Raspberry Pi.” [1]

This tiny module has some special features. Firstly, it is very easy to use, even if we are getting started with building hardware and software we will be able to use it. Secondly, it is expandable, and plugs into a variety of dock boards that increase its potential. Finally, it is affordable, it is only $5 which allows everyone to get access to it.

Figure 4.1 - Omega2 module [6].

It comes in two versions: Omega2 and Omega2+. The Omega2+ uses the same chipset as the Omega2, but has double the built-in storage and memory and it also have a MicroSD slot built in.
<table>
<thead>
<tr>
<th></th>
<th>Omega2</th>
<th>Omega2+</th>
</tr>
</thead>
<tbody>
<tr>
<td>580 MHZ CPU</td>
<td>580 MHZ CPU</td>
<td></td>
</tr>
<tr>
<td>64 MB DDR2 Memory</td>
<td>128 MB DDR2 Memory</td>
<td></td>
</tr>
<tr>
<td>16 MB Flash Storage</td>
<td>32 MB Flash Storage</td>
<td></td>
</tr>
<tr>
<td>USB 2.0</td>
<td>USB 2.0</td>
<td></td>
</tr>
<tr>
<td>MicroSD Slot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b/g/n Wi-Fi</td>
<td>b/g/n Wi-Fi</td>
<td></td>
</tr>
<tr>
<td>15 GPIO</td>
<td>15 GPIO</td>
<td></td>
</tr>
<tr>
<td>2 PWM</td>
<td>2 PWM</td>
<td></td>
</tr>
<tr>
<td>2 UART</td>
<td>2 UART</td>
<td></td>
</tr>
<tr>
<td>1 I²C</td>
<td>1 I²C</td>
<td></td>
</tr>
<tr>
<td>1 SPI</td>
<td>1 SPI</td>
<td></td>
</tr>
<tr>
<td>1 I²S</td>
<td>1 I²S</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4-1 - Comparison between Omega2 and Omega2+ [1].*

There is a wide range of digital inputs and output available on the Omega2, which allows us to interface to the external world like GPIO pins or I²C and SPI communications.
The Omega2 module has expansion “docks” boards to plug into, which can then be further expanded with a family of additional boards that Onion [6] offers. To get more information, take a look the 4.3 Omega2 Docks section and the ANNEX A.

4.2. Comparison with other modules

The potential of this module comes from his very low price compared with other products on the market with similar specifications. In Table 4.2 it is shown a comparison with some other modules that could also be used.
The main difference between the Omega2 module and its competitors is, as mentioned before, the low price combined with the tiny size and high flexibilities. For only $5 the basic Omega2 can be purchased, and the Omega2+ with double storage and memory and a microSD slot for $9.

An Arduino with similar specification is the Arduino MKR1000 WIFI, but this one breaks off with
a high price of $31. Instead of the Arduino, we can think about the C.H.I.P. This one can seem very competitive, but the only and considerable trouble it is his availability due to the large minimum order quantity (MOQ) required. If we want a device, we have to order it and wait until there is a large number of orders. This can mean to wait some months.

The only board that can face up the Omega2 is a giant best seller: Raspberry Pi Zero W. This boards come also with WI-FI connectivity. The Raspberry board is faster and has a better RAM but the Omega2 is still a competitor because its capacities are enough for our basic and medium-requirement projects. Moreover, we must talk about the connectivity to the board. While the Raspberry board has an HDMI socket, the Omega has not. This is because Omega has chosen to incorporate the Onion Cloud [9], what allows to have access to the board through any browser or any computer or platform.

To sum up we could think to use the Raspberry rather than the Omega, but we must not compare it because the Raspberry is more used as a little PC. But if we compare an Omega with an Arduino we can say that the Omega2 has nothing to be jealous of the Arduino. As a result, the Omega2 board is one of the best candidates to build an IoT device.

### 4.3. Omega2 Docks

If the idea is to build a project, we may need some dock boards for a full experience. When we start developing, there is one board we cannot be missed, the Expansion Dock. This dock has been used in the project to facilitate the connection with other boards and the power supply. But there are also more docks that can benefit our projects depending on our needs. In the report the Expansion Dock has been fully explained and analyzed. In ANNEX A we can find a description for all the Docks and Expansions available on Onion Online Store [10].

The Expansion Dock [11] is a powerful piece of hardware that simplifies the usage of our module. It allows to power the Omega and communicate with it via serial through the Micro-USB port, and makes it easier to use the GPIOs and Onion Expansions.

The Expansion Dock is powered by the Micro-USB port that supplies 5 V to the Dock. This voltage is stepped down to the required 3.3 V required to power the Omega, and also provides 5 V to the Expansions and USB Host port.
The Expansion Header is a tool that gives an easy access to the Omega’s GPIOs. The pinout diagram in Fig. 4.4 shows the Expansion Header’s connections and the possible multiplexing options:

By default, the Serial, SPI, and I2C pins implement those communication protocols and cannot be used as GPIOs. Similarly, the I2S and PWM pins are set to GPIO mode by default.

The Omega’s USB Port can be used to connect to all sorts of devices, as can be a USB storage device to extend the storage space.

The Power switch cuts the power to the Omega, but not the serial chip. This means our computer will still detect a USB serial device, but will not be able to communicate with the Omega.
And finally, the reset button on the dock is connected directly to the Omega’s Reset GPIO. By pressing can do two things: reboot (momentarily pressing), or factory restore (pressing and holding the reset button for 10 seconds).

4.4. Setup

The Omega2 is incredibly easy to get started with and use. It comes pre-loaded with a setup wizard which walks you through the quick process of connecting to our Wi-Fi network, updating to the latest firmware, and connecting to the cloud.

The very first thing we must do is to power it up. In our case, we have the Expansion Dock so we should connect the micro USB connection to a 5 V power supply. Once, we switch the power switch and wait until it boots. In about a minute, the LED will stop blinking and will remain solid. This means that the Omega has completed its boot sequence.

![Omega on the Expansion Dock](image)

Figure 4.5 - Omega on the Expansion Dock connected to the power supply [12].

Before running the setup wizard, our computer may need some extra configuration, depending on the operative system running on the computer:

- If we are using Windows, we must install Apple’s Bonjour Service [13].
- If we are using OS X, we are all set to go.
- If we are using Linux, the Zeroconf services should already be installed and we will be good to go.

What we must do next is to connect to the Omega’s Wi-Fi Network. We need to remember that the Omega is a Linux server, so it creates a Wi-Fi access point. The Wi-Fi network is named the same as your Omega and the default password is “12345678”. That password can be changed as desired.
Knowing our Omega’s name, it is very easy, we only need to look for the MAC address printed on the Omega’s sticker. As we can see in Fig. 4.5, the MAC address is: 40A36BC0 5931. For the moment, we are only interested in the last four digits that are in bold. Our Omega’s name will be Omega-ABCD, where ABCD are the last four digits from the sticker, that in our case will be: Omega-5931.

Once we are connected to the Omega’s Wi-Fi server, we are going to open a browser and navigate to http://omega-ABCD.local/. If the page does not load, we can also browse to http://192.168.3.1. We are now at the Setup Wizard and we only need to follow the instructions step by step.

4.5. Connecting to the Omega

Now that our Omega is already setup, connected to a Wi-Fi network, and updated, we can start working with it. Accessing into the Omega is easy. For that we must access to the Omega’s command line. There are three different ways:

- Using the local network to connect through SSH.
- Using a USB connection to connect to the serial terminal.
- Using the Omega’s Console in our browser.

As we want to use a wireless connection method, we are not going to use the USB connection to the serial terminal.

4.5.1. The Command-Line Interface

The command-line is a way of interacting with a computer by sending commands in the form of single lines of text. This is different from “point and click” (Omega’s Console) graphical user interfaces (GUI) found on most PC operating systems.

Command-line interfaces provide a more concise and powerful means to control a program or operating system, especially with regards to scripting (Shell Scripting, Python, etc).

4.5.2. Connecting with SSH

SSH stands for Secure Shell. It is a network protocol that creates a secure channel for communication between two devices on the same network. It can be used to secure many different types of communication, but here we will use it to login to the Omega’s command prompt. When using SSH, the Omega and the computer communicate over the Wi-Fi network.
The connection method will be different depending on what Operating System we are running on the computer. For Linux and Mac OS X we only need to open the terminal console. But, Windows OS needs one step more. We must download a terminal emulator, for example, PuTTY [14] since it is free and open-source.

While in our Linux or Mac OS X terminal we will need to write “ssh root@omega-ABCD.local”, in PuTTY (Windows OS) we will need to Configure an SSH connection to “omega-ABCD.local” on port “22”. We must remember that ABCD, in the Omega’s name means the last 4 digits from our personal Omega’s MAC address. If “omega-ABCD.local” does not works, we can try with the IP address like: “192.168.3.1”.

![PuTTY Configuration](image)

*Figure 4.6- How to fill PuTTY’s Windows [Own source]*.

When prompted, we are going to need to write the default credentials:

1. Login as / User: “root” (Only Windows OS)
2. Password: “onionier”

We are now connected. We will get something similar as in Fig. 4.7.
4.5.3. Connecting with Omega’s Console.

The Onion Console is a web-based tool (see Fig. 4.8). It is a virtual desktop that gives access to many apps that allows us to interact with the Omega in a more visual way than the command-line. We also have a terminal command line in the Console.

As we can see in Fig. 4.8, these apps can be used to configure and control the Omega and
use some expansions docks through the browser. This is all conveniently hosted on Omega and can be accessed by visiting the Omega's webpage: \texttt{http://Omega-ABCD.local/}.

To use it we firstly have to install the Console by running the following commands on the command-line:
\begin{verbatim}
uci set onion.console.setup=1
uci set onion.console.install=2
uci commit onion
\end{verbatim}

On the Omega's next reboot, the Console will be installed automatically.

As initializing the SSH, we must enter, when prompted, the default login info:

- Username: “root”
- Password: “onionier”

We will need to be on the same network as your Omega in order to access the Console or being connected to the Omega’s Wi-Fi. It is recommend connecting the Omega to our local Wi-Fi (house or office Wi-Fi) and connect our PC to the Omegas Wi-Fi. This way the Omega will act as a bridge, allowing the PC to have internet access and being able to connect to the Console.

If the previous method does not work, we can try writing the Omega’s IP Address, \texttt{http://192.168.3.1/}. Knowing our Omega’s IP Address, it is as easy as writing on the command line: “ifconfig”. The IP Address will be displayed after “addr:”, for example, “addr:192.168.3.1”

\section*{4.6. Using/developing with the Omega}

The Onion Omega2, as mentioned earlier, is a Linux server, what means that it can support every programming language. By installing the appropriated library, the Omega will run and compile any program we desire. Python has been the programming language chosen for the development of the current project. Many reasons justify this chosen option, but the main reason is that we wanted to use the same programming language learned during the degree.

Using Python on our Omega is very simple. We only need to install the package. We can choose between the 2.70 3.0 version on Python and the \textit{light} or \textit{full} version. The package that we have chosen for the project is the light 2.7 Python version to save space since the light version has 3.8 MB less and can be installed by writing in the command line: “\texttt{opkg install python-light}”.

If we want to start developing and programming, we must know a few things before. The
module is arranged in files and directories. An advice is to always place our projects in the /root directory. This directory is the best place to ensure that our files don't get deleted when we update the firmware.

The /root directory is not the only directory. If we are in a Windows computer, by running PuTTY, we get directly into /root directory. However, in Linux or Mac OS, we get into /, the top directory. In this situation to know in which directory we are, we must type “pwd” (print working directory). So, if as we have recommended, we want to go to the /root directory, we must type “cd root”. Before getting into the directory we should have check that the directory is in the current directory. Knowing what is in our current directory is as simple as typing “ls”.

![Figure 4.9 – Examples of commands [12].](image)

There are more useful commands like “cd ..” which will get you to one level to the parent directory. We have also “mkdir” to create a directory or “rmdir” to remove a directory. For more commands see references [12] and [15].

Once we know better how to use the command-line, it is time to start coding. But here it is where we are going to switch to the Console. We hardly recommend using the console editor. We can use the default editor vim but it is not very user friendly and can get confused if we are not used to. That is the reason we are going to explain a little bit how to use the Console’s editor.
We have already seen the Console and how to get in. In Fig. 4.8 we have the Console’s windows where we can see the *editor* icon. If we click-in, we will see something like in Fig. 4.10.

![Editor's windows in Console](image)

*Figure 4.10 - Editor’s windows in Console [Own source]*

On the left side of the windows (Fig. 4.10) there are several directories. On the top we have “/”, which means that we are in the top directory. On the top-right side, there are the *Upload File, New Folder* and *New File* icons. Those icons are the “point and click” substitute to the commands seen earlier. If we already have a coding file and we just want to put into the Omega, we have to use the Upload File icon.
Figure 4.11 is the windows for the /root/scripts directory. This directory is where we have our project files. We must remark some difference in this new window. On the top-right we have new icons which are: Delete Folder, Delete File, Rename Folder, Rename File and Save. We must say that they do not have any shortcut as we could think such as CTR+S to save.

A last new thing to remark is in the blue box. We have a tab way to switch between the already opened files. We also miss some mark in the tab letting us now that there is a new change from the last time the file was saved.

As we have seen, the Console’s editor has a lot to improve but is still a better and recommended way to edit our project files.
5. Robotic Arm

In order to use the Omega module in a real project, we decided to continue increasing the potential of a current project. We are talking about a rolling and incremental project focused on a Robotic Arm.

This project has been designed and built by Guillem Ferré, programmed by Javier Estévez and improved by Alejandro Antón in their respective End of Bachelor Degree [5], [4] and End of Master Degree [2]. The first two students created a full working small scale robotic arm, as shown in Fig. 5.1.

![Figure 5.1 - Robotic arm build by Guillem and Javier [2]](image)

Besides these two projects, Alejandro Antón developed a great project based on the robot by implementing a huge number of improvements and ending with a new and better robot. He changed some constructive pieces that were broken, he replaced some servo motors by a more torque power servomotors, he changed the base tread and he also added a camera and a hand based on tweezers.

The ending robotic arm, belonging to the Department of Electronic Engineering in ETSEIB, was a Raspberry based robot who sent PWM data to the servomotors through the PCA9685 board. As we can see in Fig. 5.2, the Raspberry sends data using the I2C bus to the PCA9685 board, that at the same time, sends a PWM control command to each of the 5 servomotors. The inputs were the image captured by the camera and the user desired position sent by a mobile app through a KIVY server. In this new version for the robot, the camera will be removed as it is only Raspberry compatible. We will also change the APP and make a new web-based
API for an interactive control of the robot.

![Old project diagram](image)

**Figure 5.2 - Old project diagram [Own source].**

The aim of the project was not to make any improvement on the robots besides changing the control module. But, during this time we have been forced to change a structural part that was broken (Fig. 5.3) for a new one (Fig. 5.4) also 3D printed at the RepRap room [16] at ETSEIB.

![Broken part](image)

**Figure 5.3 - Broken part**

![New 3D printed part](image)

**Figure 5.4 - New 3D printed part**

In order to understand how this robot works, and investigation was required about the servomotors, PWM signals and I2C protocols. Therefore, this is going to be summarized in the following sections.

### 5.1. Servomotor & PWM signal

The current motors on the robot are servomotors. This type of motor is controlled by his
position through PWM pulses. Servos, short way to call them, have a rated control voltage between 4.8 and 7.2 V with an electric current between 0.2 and 5 A.

The value of the voltage rating will depend on how many servos will be connected at the same time. Servomotors present an angular movement in between 0º and 180º normally. If we want a largest range on position, we are going to need a continuous rotation servo. Those motors are controlled by speed despite the servomotors that are controlled by their position.

Servos can be analog or digital, although most of them are analog, like the current 5 ones on the robot. The analog and digital servomotors only differ in the way the pulse signals are sent to the motor. The analogs have a small control circuit while the digital ones have a small microcontroller. Digital servomotors are more accurate and tend to be of better quality and with a longer service life. They can also work at higher frequencies (in the order of 300 Hz), but they are more expensive and consume more power. On the other hand, analog servomotors are simpler and cheaper, but they can work properly for simple low power applications. They only work at a frequency of 50 Hz and have low consumption.

![Figure 5.5 – Generic servomotor](image)

To control the servomotors, we need to use PWM pulses. A PWM pulse is defined by a Duty Cycle and a fixed frequency. The Duty Cycle is a parameter [0-1] that indicates the time when the signal is a high level, in percentage. On the other hand, the frequency indicates the number of pulses send by second, normally at 50 Hz.

Each servo has some characteristic pulses that depend on the angular position, as well as the model and the manufacturer. These pulses are those that correspond to the extreme positions of the movement, 0º and 180º for example.

Some servomotors indicate the pulse that corresponds to the minimum and maximum values of their movement in their datasheet. As shown in Figure 5.4, the servomotor is controlled with pulses between 1 and 2 ms, corresponding this positions to 0 and 180 degrees, respectively. If you want a position between the two values, you must apply a simple interpolation between
Despite knowing these theoretical values provided by the manufacturer, it is advisable to find out the actual values of servomotors experimentally. The following Table shows the angular ranges and pulse ranges of each servo and in which channel they are plugged in.

<table>
<thead>
<tr>
<th>Name</th>
<th>Programming Name</th>
<th>Angular limits [º]</th>
<th>Pulses limits</th>
<th>Canal at PCA6985</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation</td>
<td>“Rotacio”</td>
<td>[0, 180]</td>
<td>[0, 580]</td>
<td>0</td>
</tr>
<tr>
<td>Axis</td>
<td>“Eix”</td>
<td>[0, 125]</td>
<td>[168, 487]</td>
<td>1</td>
</tr>
<tr>
<td>Crank</td>
<td>“Manovella”</td>
<td>[95, 180]</td>
<td>[400, 200]</td>
<td>2</td>
</tr>
<tr>
<td>Head</td>
<td>“Gircap”</td>
<td>[0, 180]</td>
<td>[120, 485]</td>
<td>4</td>
</tr>
<tr>
<td>Hand</td>
<td>“Pinces”</td>
<td>[0, 69.75]</td>
<td>[320, 468]</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5-1 Servomotors values. [2]

5.2. I2C

The communication between the Omega and the PCA9685 it is done by the I2C bus. I2C or
I2C, Inter-Integrated Circuit, protocol [19] is a protocol allows multiple “slave” to communicate with one or more “master”. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.

![Diagram of a two master and two slave wire connection](image-url)

**Figure 5.7 - Representation of a two master and two slave wire connection [19].**

I2C requires only two wires and those two wires can support up to 1008 slave devices. I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can’t talk to each other over the bus and must take turns using the bus lines).

Each I2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master but some slaves may force the clock low at times to delay the master sending more data.

### 5.2.1. Onion I2C Python Module

The onionI2C Python module in the OmegaExpansion package provides a Python object that serves as a wrapper around the libonionI2C [20], a C library. The Onion I2C library uses the /dev/i2c-0 adapter, and implements read and write functions with I2C devices.

To install the Python module, we need to run the following commands:

```
opkg update
opkg install python-light pyOnionI2C
```

This will install the module to /usr/lib/python2.7/OmegaExpansion/.

To add the Onion I2C Module to our Python program, we just need to include:

```
from OmegaExpansion import onionI2C
```

The Python module has some specific functions. The following Table shows these 5 functions:
Function | Prototype |
---|---|
Initialization | onionI2C.OnionI2C() |
Reading Bytes | readBytes(devAddr, addr, size) |
Write a Single Byte | writeByte(devAddr, addr, value) |
Write a List of Bytes | writeBytes(devAddr, addr, values) |
Write a List of Bytes without Specifying an Address | write(devAddr, values) |

**Table 5-2 Python module functions [12].**

The “Initialization”, “Reading Bytes” and “Write a Single Byte” are the only functions that will be used forward. To get more information about how do we use the other functions, visit the Onion documentation [12].

- **Initialization - onionI2C.OnionI2C()**

The object needs to be initialized before it can be used for reading and writing:

```python
i2c = onionI2C.OnionI2C()
```

After this call, the object can be used for reading and writing.

The constructor has an optional argument that defines the I2C device adapter number. This corresponds to the X in `/dev/i2c-X`.

If no argument is supplied, the adapter will be set to `/dev/i2c-0`. This is the only default adapter on the Omega and should suit most use cases.

If a specific case requires a different adapter, we only need to add an integer argument to the constructor call.

- **Reading Bytes - readBytes()**

This function reads a specified number of bytes from a specific device on the I2C bus, and returns them in a list:

```python
valList = i2c.readBytes(devAddr, addr, size)
```
The `devAddr` argument defines the I2C slave device address. The `addr` argument defines the address on the device from which to read. And the `size` argument is the number of bytes to be read.

- **Write a Single Byte - writeByte()**

This function will write a single byte to a specific device on the I2C bus:

```python
status = i2c.WriteByte(devAddr, addr, value)
```

The `devAddr` and `addr` arguments are the same as the reading byte function. The `value` argument stands for the value of byte to send.

### 5.3. PCA9685

The PCA9685 board is a PWM controller with 16 channels that works through an I2C bus. This board was designed to control LEDs, but it can be also used to control servomotors because they both can be controlled by PWM signals.

![PCA9685 board](image)

*Figure 5.8 - PCA9685 board [21].*

This board is very useful if we are working with a huge number of servomotors because it allows to work up with 16 units with only 4 signals. If we had to control the 5 servomotors without the board, up to 15 signals would be necessary. Instead, with the PCA board, only 4 signals are needed.
The board is connected to 4 signals, which are:

1. Power supply 3.3 V: **VCC**
2. Ground signal: **GND**
3. Clock signal: **SCL**
4. Data signal: **SDA**

The 3.3V **VCC** signal is only needed for the control process. That is why we are connecting the board to an extra couple of signals, 5 V **VCC** and **GND**, which will give the necessary power to each motor. The wiring connection of the board with the Omega board and the servos is shown in Fig. 5.10.

![Figure 5.9- Wire representation made with Fritzing [Own source].](image)

The board allows to control individually each motor with a 12-bit resolution and a large range of frequencies. As we have said before, we need to know the starting and ending pulses and the working frequency. For example, if we have a servo with:

- Minimum and maximum pulses: 1 and 2 ms (0 and 180º).
- 12 bits resolution: \(12^2=4096\) values.
- Period of 20 ms (50 Hz).

With an easy equation we have that the pulse corresponding to 180º position is:

\[
\frac{2\text{ ms}}{20\text{ ms}} \cdot 4096 = 409.6
\]

This theoretical pulse width of 406.9 (rounded up to 407) is the one sent by the PCA board through a channel “n” to move to the 180º position. The movement will be sharp and abrupt so
a control algorithm will be designed in the programming phase.

5.3.1. PCA Library

The PCA9685 has its own library to control the servos. This software allows the user to select the desired frequency and sending PWM signals to each servo. The problem I have found was that this library was only available for Raspberry [22] or Arduino [23]. Consequently, a new challenge came up.

This seemed very difficult at the beginning. When a library is downloaded and used, we normally not need to understand it, we just use it. But when a library cannot be founded, this means that nobody has done it before, and therefore it is a challenge.

We could not use the Raspberry or Arduino libraries because they import their own (Raspberry or Arduino) I2C library. So, I decided to adapt one of them to the Omega2. We chose the Raspberry because it was already coded in Python. A laborious investigation was made understanding the code. But, finally, a with a few changes the library was ready to be used on the Omega2.

The library is composed by a primary part where we have some imports and some parameters definition. This part had not been modified. Then, there is a the PCA9685() class with 4 methods. The new driver code is on ANNEX B.

- class PCA9685(object):

  [1] __init__ (self, address=PCA9685_ADDRESS, i2c=None, **kwargs):

On the new driver, we import the onionI2C library from the OmegaExpansion. Then an instance of the library is created. We must remark that the way to set up the instance is different. As seen in 5.2.1 Onion I2C Python Module, to create a new instance we must write: name_instance = onionI2C.OnionI2C() while on the Raspberry code, it has to be done in two steps: i2c = I2C and self._device = i2c.get_i2c_device(0x00 , **kwargs).

Another thing changed is the write8 function. Reading the Adafruit Python I2C library [24], we observe that this function is to “Write an 8-bit value on the bus (without register)”. There is no need to specify the register address because it is defined when the I2C instance is created. But, when we create the I2C instance on Omega2 we do not specify the I2C slave device address. Therefore, to send a single byte, we use the writeByte(devAddr, addr, value) function where we specify the slave device address in the devAddr argument.
Finally, the last thing changed is the `readU8` function. According to [24], the `readU8` function "reads an unsigned byte from the specified register". To read a single byte on the new drive we must use the Omega I2C Python function: `readBytes(devAddr, addr, size)`. Only a byte has to be written so the size parameter has to be a "1".

The highlighted parts in the code are those that had been added or changed. To success in this thesis part, the PCA9685 datasheet [25] was used.

```
if i2c is None:
    import Adafruit_GPIO.I2C as I2C
    i2c = I2C
self._device = i2c.get_i2c_device(address, **kwargs)
self.set_all_pwm(0, 0)
self._device.write8(MODE2, OUTDRV)
self._device.write8(MODE1, ALLCALL)
time.sleep(0.005)
mode1 = self._device.readU8(MODE1)
mode1 = mode1 & ~SLEEP
self._device.write8(MODE1, mode1)
time.sleep(0.005)

self.address = address
if i2c is None:
    from OmegaExpansion import onionI2C
    self.i2c = onionI2C.OnionI2C()
self.set_all_pwm(0, 0)
self.i2c.WriteByte(self.address,MODE2, OUTDRV)
self.i2c.WriteByte(self.address,MODE1,ALLCALL)
time.sleep(0.005)
model = self.i2c.readBytes(self.address,MODE1,1)
model = model[0]
mode1 = mode1 & ~SLEEP
self.i2c.WriteByte(self.address,MODE1,model)
time.sleep(0.005)
```


This method sets the desired PWM signal frequency. In this method, only the `readU8` and `write8` functions have been changed. An extra little change has been made because the `readBytes` function returns a list with the bytes so a "[0]" has been added in line 8 to refer to the byte from the `readBytes` function.

```
prescaleval = 25000000.0  # 25MHz
prescaleval /= 4096.0     # 12-bit
prescaleval /= float(freq_hz)
prescaleval = 1.0
prescale = int(math.floor(prescaleval+0.5))
oldmode = self._device.readU8(MODE1);
newmode = (oldmode & 0x7F) | 0x10
self._device.write8(MODE1, newmode)
self._device.write8(PRESCALE, prescale)
```

```
prescaleval = 25000000.0  # 25MHz
prescaleval /= 4096.0     # 12-bit
prescaleval /= float(freq_hz)
prescaleval = 1.0
prescale = int(math.floor(prescaleval + 0.5))
oldmode = self.i2c.readBytes(self.address, MODE1,1)
newmode = (oldmode[0] & 0x7F) | 0x10
self.i2c.WriteByte(self.address,MODE1,newmode)
```
Control of a robotic arm using an Omega2+ module

```python
self._device.write8(MODE1, oldmode)
time.sleep(0.005)
self._device.write8(MODE1, oldmode | 0x80)
```

[3] `set_pwm(self, channel, on, off):`

This method sends a PWM signal to the channel specified. In this method, only the `write8` function has been changed. We must remark that the parameter `on` and `off` had been transformed to integer due to an error when the code was executed.

<table>
<thead>
<tr>
<th>Old Code</th>
<th>New code</th>
</tr>
</thead>
<tbody>
<tr>
<td>self._device.write8(LED0_ON_L+4*channel, on &amp; 0xFF)</td>
<td>self.i2c.WriteByte(self.address, LED0_ON_L+4*channel, int(on) &amp; 0xFF)</td>
</tr>
<tr>
<td>self._device.write8(LED0_ON_H+4*channel, on &gt;&gt; 8)</td>
<td>self.i2c.WriteByte(self.address, LED0_ON_H+4*channel, int(on) &gt;&gt; 8)</td>
</tr>
<tr>
<td>self._device.write8(LED0_OFF_L+4*channel, off &amp; 0xFF)</td>
<td>self.i2c.WriteByte(self.address, LED0_OFF_L+4*channel, int(off) &amp; 0xFF)</td>
</tr>
<tr>
<td>self._device.write8(LED0_OFF_H+4*channel, off &gt;&gt; 8)</td>
<td>self.i2c.WriteByte(self.address, LED0_OFF_H+4*channel, int(off) &gt;&gt; 8)</td>
</tr>
</tbody>
</table>

[4] `set_all_pwm(self, on, off):`

This method sends PWM signal to all the channels. In this method, only the `write8` function has been changed. We must remark that the parameter `on` and `off` had been transformed to integer due to an error when the code was executed.

<table>
<thead>
<tr>
<th>Old Code</th>
<th>New code</th>
</tr>
</thead>
<tbody>
<tr>
<td>self._device.write8(ALL_LED_ON_L, on &amp; 0xFF)</td>
<td>self.i2c.WriteByte(self.address, ALL_LED_ON_L, int(on) &amp; 0xFF)</td>
</tr>
<tr>
<td>self._device.write8(ALL_LED_ON_H, on &gt;&gt; 8)</td>
<td>self.i2c.WriteByte(self.address, ALL_LED_ON_H, int(on) &gt;&gt; 8)</td>
</tr>
<tr>
<td>self._device.write8(ALL_LED_OFF_L, off &amp; 0xFF)</td>
<td>self.i2c.WriteByte(self.address, ALL_LED_OFF_L, int(off) &amp; 0xFF)</td>
</tr>
<tr>
<td>self._device.write8(ALL_LED_OFF_H, off &gt;&gt; 8)</td>
<td>self.i2c.WriteByte(self.address, ALL_LED_OFF_H, int(off) &gt;&gt; 8)</td>
</tr>
</tbody>
</table>
5.4. Robot mechanism

The robot mechanism can be split in two different movements. On one hand, there is the main mechanism which is very important to be able control the position of the head. On the other hand, there is the analysis of the hand mechanism. We must remark that no improvements were made in this section, it is only a summary from Guillem [5] and Alejandro’s [2] thesis.

5.4.1. Main mechanism

In this section, I am going to summarize the design and mechanic analysis as well as the mathematic solution of the position.

As mentioned before, the design was made by Guillem [5] during his Final Degree project (TFG). The main mechanism is made up by a main arm and a crank. By controlling the angular position of those two elements, controlled by $\varphi_1$ and $\varphi_2$ respectively, we can move, in a two-dimension plane, the head position, represented with the point “P” (Fig. 5.11). Fig 5.12 and Fig.13 show the schematic representation of the structure.

Figure 5.10 - Available positions in a 2D plane [5].
The equations describing the movement, obtained in [5], are:

\[ P_x = l_2 \cdot \cos \varphi_1 + l_1 \cdot \cos \varphi_4 + l_2 \cdot \cos 35° \]  \hspace{1cm} (Eq. 5.1.)

\[ P_y = l_2 \cdot \sin \varphi_1 + l_1 \cdot \sin \varphi_4 + l_2 \cdot h \]  \hspace{1cm} (Eq. 5.2.)
\[ l_2 \cdot \cos \varphi_1 = l_3 \cdot \cos \varphi_4 + l_3 \cdot \cos \varphi_2 + l_2 \cdot \cos \varphi_3 \]  
(Eq. 5.3.)

\[ l_2 \cdot \sin \varphi_1 = l_3 \cdot \sin \varphi_4 + l_3 \cdot \sin \varphi_2 + l_2 \cdot \sin \varphi_3 \]  
(Eq. 5.4.)

Value “h” stands for the height between O₁ and the base. This set coordinates origin under O₁ at the table’s height. The value of h is: 145 mm.

(Eq. 5.1.) and (Eq. 5.2.) describe the P position by a configuration of \( \varphi_1 \) and \( \varphi_2 \). (Eq. 4.3) and (Eq. 4.4.) connect the 4 angles that describe the movement. A combination of those equations is the mathematic solution to locate the point P, the main objective. But the inconvenient is that we have 4 equations for 6 unknown values, what seems incompatible. In our case, we will always know 2 of the 6 values. In the case we know the P position, \( P_x \) and \( P_y \) values, we will get the combination of the main arm and the crank angles, \( \varphi_1 \) and \( \varphi_2 \) respectively, and the values of \( \varphi_3 \) and \( \varphi_4 \), which we will use to check the validity of the results. In contrast, we can also know the angles of the servos \( \varphi_1 \) and \( \varphi_2 \) we have, so we will get the P position, \( P_x \) and \( P_y \) values, and the values of \( \varphi_3 \) and \( \varphi_4 \). In the first case, where we know the P position, it is very simple to solve by only two steps (solved in [5]). Firstly, we solve the combination of (Eq. 5.1) and (Eq. 5.2.), knowing \( P_x \) and \( P_y \) and getting \( \varphi_1 \) and \( \varphi_4 \). Secondly and finally, we solve the combination of (Eq. 4.3) and (Eq. 4.4.), knowing \( \varphi_1 \) and \( \varphi_4 \) and getting \( \varphi_2 \) and \( \varphi_3 \). In the second case, where we know the position of the main arm and the crank, \( \varphi_1 \) and \( \varphi_2 \) respectively, it is as simple as the first case to solve (solved in [5]). Firstly, we solve the combination of (Eq. 5.3) and (Eq. 5.4.), knowing \( \varphi_1 \) and \( \varphi_2 \) and getting \( \varphi_3 \) and \( \varphi_4 \). Second and finally, we solve the combination of (Eq. 4.3) and (Eq. 4.4.), knowing \( \varphi_1 \) and \( \varphi_4 \) and getting \( P_x \) and \( P_y \).

Once we know the mathematic solution for the movement, this has to be implemented. Alejandro, in [2], came up with an analytic solution that can be written in coding lines. Starting with the following generic parameters system:

\[ P_1 = d_1 \cdot \cos \beta_1 + d_2 \cdot \cos \beta_2 \]  
(Eq. 5.5.)

\[ P_2 = d_1 \cdot \sin \beta_1 + d_2 \cdot \sin \beta_2 \]  
(Eq. 5.6.)

After some operations we get:
\[
\begin{align*}
\left\{ \begin{array}{l}
\beta_{2,i} = \cos^{-1} a_i \; \text{for} \; 1 \leq i \leq 2 \\
\beta_{2,(2+i)} = -\cos^{-1} a_i \; \text{for} \; 1 \leq i \leq 2
\end{array} \right. \\
\text{(Eq. 5.7.)}
\end{align*}
\]

\[
\begin{align*}
\left\{ \begin{array}{l}
\beta_{1,i} = \sin^{-1} \left( \frac{1}{d_i} \cdot [P_2 - d_2 \cdot \sin \beta_{2,i}] \right) \; \text{for} \; 1 \leq i \leq 4 \\
\beta_{1,(4+i)} = \pi - \sin^{-1} \left( \frac{1}{d_1} \cdot [P_2 - d_2 \cdot \sin \beta_{2,i}] \right) \; \text{for} \; 1 \leq i \leq 4
\end{array} \right. \\
\text{(Eq. 5.8.)}
\end{align*}
\]

From (Eq. 5.7.) we get the 4 possible values for $\beta_2$. Using those 4 possible values and (Eq. 5.8.) we finally get the 8 pair of possible $\beta_1$ and $\beta_2$. It is worth mentioning that some of those solutions will not be possible due to invalid values of sinus or cosine. Once the mathematical impossibilities are discarded, we also have to discard those solutions that do not fit with physical criteria like out of range values. For the detailed solve development see [2].

### 5.4.2. Hand mechanism

The main mechanism is not the only mechanism present in the robot, since Alejandro introduced a hand making possible to grab things. This hand is made up by two gear wheels controlled by a servo. By a symmetric mechanism we can transform a rotating movement into a linear movement. Figure 5.14 shows the assembled mechanism. The left wheel is the one that transmits the torque power from the servo to the mechanism.

![Figure 5.13 - Image of the hand. [Own source]](image)

What we expect to know is the geometric relation between the angular position for the servo and the hands gap. Another relation to analyses is the height where the hands can grab the object. The servo position is represented by $\phi$ the gap by $s$ and the height by $h$. 
Figure 5.14 - Schematic representation of the hands mechanism. [2]

The real values for the parameters shown in Figure 15 are:

\[ \begin{align*}
  l_1 & = 26 \\
  e & = 15 \\
  b & = 6 \\
  l_3 & = 23 \\
  a & = 19 \\
  d & = 24 \\
  ch & = 1.5 \\
  c & = 26 \\
  cv & = 12
\end{align*} \]

All those values are in mm. The value \( ch \) stands for an extra pad for a better grip.

The relations between the interest parameters with the real values are shown in the following equations:

\[ s = 2 \cdot l_1 \cdot \cos \varphi + d - 2(e + ch) - 2b = 52 \cdot \cos \varphi - 18 \]  
\[ (Eq. 5.9. \) 

\[ \varphi = \cos^{-1} \left( \frac{s-d+2(e+ch)+2b}{2 \cdot l_1} \right) = \cos^{-1} \left( \frac{s+18}{52} \right) \]  
\[ (Eq. 5.10. \) 

\[ h = l_1 \cdot \sin \varphi + c + a + l_3 = 22 \cdot \sin \varphi + 68 \]  
\[ (Eq. 5.11. \) 

We can find the maximum value, which will be useful for the programming development of the
software, for the standing parameters from the equation, namely:

\[ s_{\text{max}} = s(\varphi_{\text{suposed}} = 22^\circ) = 52 \cdot \cos 22 - 18 = 30.21 \text{ mm} \]  
Eq. 5.12.

\[ \varphi_{\text{max}} = \varphi(s = 0) = \cos^{-1}\left(\frac{18}{52}\right) = 69.75^\circ \]  
Eq. 5.13.

\[ h_{\text{max}} = h(\varphi_{\text{max}}) = 22 \cdot \sin 69.75 + 68 = 92.39 \text{ mm} \]  
Eq. 5.14.
6. Software

The software is the section where all the files structure and the software code are going to be explained. This part is where the main differences from Alejandro’s project [2] are. A lot of significative modifications have been made like recoding the some classes or even the development of a new controlling interface.

6.1. Description of the global architecture

The global software architecture is made up by an API and the Omega's server. The Web-Based API plays a client role while the Omega plays a server role. This communication between the user interface and the robot is different from [2]. While in Alejandro's project the communication was based in the Kivy server and a Flask server, now this communication is between the WebSocket Server [26] and the Omega’s Server. This bidirectional communication will be elaborately explained in section 6.4 Web-based APP. Fig. 6.1 is a representation of the main interactions in the global architecture.

![Figure 6.1 - Schematic representation of the global architecture [Own source]](image)

The user interacts with the web-based API. This interaction triggers the message-based communication between the WebSocket's server and the Omega’s server. This communication is bidirectional, what means that the API will also receive information from the Omega. Once the message arrives to the Omega’s server, this one will run a specific part of the Python program.
This interaction is the only input in the global architecture while in the previous project the robot had also a camera on the head. This camera could not be incorporated to the current project since it is only compatible with Raspberry.

6.2. How to run

Starting to use the robot has previous tasks to be done. The API which basically sends messages (“strings”) to the port 8888 in Omega’s server. Therefore, we must run the robot.py file because is where the communication is opened. This means that if we want to use the API in a tablet or phone we will also need to use the PC to execute the robot.py file on the command-line. By following the steps, we will be able to control the robot from the API in the platform we prefer (PC/Smartphone/Tablet). It can seem laborious but when we are used to it, it takes less than 1.5 minutes.

Steps:

1. Open the command line (Console/PuTTY/Terminal).
2. Once we are in the /root directory, navigate to the directory where the file robot.py is. In our case, the file is in: /root/scripts/robot.py. We only need to go to the /scripts directory by typing: cd script/.
3. To run the robot.py file we must write: python robot.py
4. While the code is running, we must to open the API. To achieve it, we must go the folder that contains the API in our PC, phone or tablet and open the html file in a browser.

5. When the message “Websocket Opened” is printed in the shell, this will mean that the

Figure 6.2 - Step 4: Open the html file in a browser [Own source].
Websocket communication will be opened. This can take up to a minute.

6. Now we are ready to control the robot.

![Figure 6.3](image)

Figure 6.3 - Capture of the shell with the steps described [Own source]

Closing it is as simple as interrupting the running code and closing the API. To interrupt the code, we must type “CTRL+C” on the shell.

If the message “Websocket Closed” is printed on the shell, it will mean that the communication is closed. This can be produced if we close the browser. We must consider that if we refresh the API page, the communication will be closed and reopened immediately. So, we will see the message “Websocket Closed” followed by a “Websocket Opened” on the shell.

During the testing phase, I have noticed that sometimes, when we have too much time the communication opened, it can be interrupted. This will mean to reboot the Omega and restart all the process. This issue is something I have not understood and is neither explained on the Websocket web [26].
6.3. Main program

The main program is a set of Python files located in the Omega. This files and structure are based on Alejandro’s project. As we can see in Fig. 6.4. I have deleted some files. This is due to the new communication system requirements, the camera being removed and the new specifications for the control system.

![Diagram of Alejandro's files structure](image)

Figure 6.4 - Alejandro's files structure [2].

The main structure has been saved. In the current files structure, Fig. 6.5, there is a top file, `robot.py`, different from the Alejandro’s. This file is the one that has to be executed as we have seen. The files been involved with the camera as `visio.py`, `view.py` and `camera.py` have been deleted because I was no longer using the camera because it was not Omega2 compatible. The remaining file deleted, `rutes.py` and `pinces.py` are no longer useful to the new requirements of the controlling system.

In Fig. 6.5, under `robot.py`, in blue, we have the Python files where we have the main classes and functions. The purple box is the PCA9685 driver, that is used in `baseservos.py`. And finally, we have the two JSON files where we have the servo information and positions.
Let's see a brief explanation of each file:

- **robot.py**: File completely new. File that contains a class who listens to the messages coming from port 8888 in Omega’s server and that can also send messages back. It is in charge to execute different methods from other files depending on the message content.

- **baseservos.py**: File that contains the BaseServos’s class. It has the assistance and preparatory functions for the Servos’s class. This class has important functions such as the one that initialize the PCA or the one that sends the PWM signal to the servos. Here the Mecanisme’s class and the Pinces’s class are initialized too, located at mecanisme.py and pinces.py respectively. This file has been lightly changed. Some methods in the class has been deleted.

- **servos.py**: File that contains the Servos’s class, which includes the different functions of movement of servomotors. The Servo’ class inherits from Baseservos’s class. The Servo’s class has the main functions that will be invoked by robot.py. In this class some new methods have been coded.

- **mecanisme.py**: File that contains the Mechanism’s class, which includes all functions necessary for the analysis of the main mechanism. Class in charge of solving the
equations described in 5.4 Robot mechanism. This file is practically the same in Alejandro’s project.

- **interpolaciones.py**: File that contains the interpolacio function, which includes the different interpolations types to be applied to a movement. There are three different possible interpolations: simple, linear or Bezier. At the same time, there are single functions for each interpolation. For each type, it generates a list with the different durations (delays) to be applied to the movement. Each delay is applied between pulses sent to a servomotor. This file is practically the same in Alejandro’s project.

- **PCA9685.py**: It is the driver in charge to send PWM signals to the servos. As we have explained in 5.3.1 PCA Library, it is an adaptation of the Raspberry driver [22]. This file contains the PCA9685’s class. It is needed by the Omega to be able to send data to the servos through the PCA board. It initializes the communication. It sets up the required frequency for the PWM and send the pulses.

- **infoservos.json**: File that contains the information of each servo. The Pulse Range, Angles Range, PCA Channel and the Duration of the movement is the information we have for each servo. This file is practically the same in Alejandro’s project. Some adjustments have been made on the servo’s time.

- **angles.json**: File that contains the actual position of each servo. It is refreshed every moment that we send a pulse to a servomotor.

Analyzing and understanding these files requires times. During this project, this time has been quite important because it took few weeks to understand each line of the code. Knowing the meaning of each method is very important when we want to adapt it to new requirements.

Facilitate time and efforts to whoever want to use this code it is one of the objective described at the beginning. This is why next pages will be dedicated to explaining each file in detail. New function will be highlighted in a different color, to beef up the contributions during this project. All the Python files have been included in ANNEX B.

### 6.3.1. robot.py

This file is completely new. It is based on a code described on a blog’s page: Control An Onion Omega2 IoT Device With Websocket Communication [27].
• Imports: `tornado.ioloop`, `tornado.websocket` and `servos`.
• `if __name__ == "__main__":` This conditional request is executed each time we run the file. Those 4 code lines are very important because there are in charge to open the 8888 port from the Omega’s server. This port will be ready to “listen” the messages coming from the Websocket server.

```python
application = tornado.web.Application([ (r"/", IoTWebsocket),
                                        application.listen(8888)
                                    tornado.ioloop.IOLoop.current().start()
```

• Class: `IoTWebsocket(tornado.websocket.WebSocketHandler)`.
• Functions in `IoTWebsocket’s Class`:

  [1] `check_origin(self, origin)`: Method overrided to return True to accept all cross-origin traffic.

  [2] `open(self)`: Invoked when a new WebSocket is opened. The message "Websocket Opened" is printed on the shell to know that the Websocket is opened. The `posicioInicialServos` function is invoked to set the robot to the reset position each time the Websocket communication is opened.

  [3] `on_message(self, message)`: Invoked when a new message arrives. The message has to be strictly a string. Therefore, I have chosen a prefix character to indicate which button has been pressed or slider has been moved. Depending on the message starting character, a different method will be executed. Prefixes are:

     • `message[0] = 0` for a rotation slider movement.
     • `message[0] = 1` for an axis slider movement.
     • `message[0] = 2` for a crank slider movement.
     • `message[0] = 4` for a head slider movement.
     • `message[0] = 5` for a hand slider movement.
     • `message[0] = 6` for a vertical slider movement.
     • `message[0] = 7` for a horizontal slider movement.
     • `message[0] = 8` for a reset button pressing.
     • `message[0] = 9` for an automatic button pressing.
Every slider movements mean an update of a single or various servos position. Therefore, messages with prefixes between 0 and 7 (included), that means that a slider has been updated, have also the new position on the message right after the prefix. The massage is: “prefix [0:7]” + “new slider value”.

Buttons do not need extra information than the prefix to signal which one has been pressed. Buttons will only make to run a single function: resetServos() or automatic().

When the user moves the axis or crank slider the P position changes. This is straightly related with the vertical and horizontal sliders. When the user moves the axis or crank slider the actualizarSlidersDesplacament() is executed. If this new position is a valid position, moureServo() will be executed, moving the servo to the new position. Either wise, a coded message is sent to the API (write_message(mes)). Depending on the code of the message, the interface will update the slider or it will show an “invalid position” message.

Vertical and horizontal sliders movement also mean an update of the P position. Therefore, when a vertical or horizontal slider is updated, the incoming message will run actualitzarSlidersServos(). In this case, it will also be sent a coded message to the API.

Fig. 6.6 is an event diagram of the on_message() method. This diagram explains in a visually way what this method does.
Figure 6.6 - Events diagram for on_message() method [Own source].
on_close(self): Invoked when the WebSocket is closed. We use it to print a message that informs that the server it is closed. The message "Websocket Closed" is printed on the shell to know that the Websocket communication is closed.

6.3.2. baseservos.py

- Imports: PCA9685, pi from math, loads and dumps from json, system from platform, time and Mecanisme’s class from mecanisme.
- Class: BaseServos.
- Functions in BaseServo’s Class:

1. __init__(self): Executed every single time an instance of the BaseServo class is initialized. Calls 4 functions: carregarMecanismes, definirParametres, inicialitzarParametres, inicialitzarPCA.

2. carregarMecanismes(self): Creates an attribute of the Mecanisme class.

3. definirParametres(self): Defines some attributes of interest like the dictionary self.anglesReset, some interpolation constants or the frequency of the PWM signals.

4. inicialitzarParametres(self): Loads the “angles.json” and “infoservos.json” files. Creates some other attributes of interest like self.dades, self.resetX or self.resetY.

5. inicialitzarPCA(self): Creates the self.pwm attribute used by the PCA driver and assign the PWM signal frequency.

6. actualitzarArxiuAngles(self): Updates the file “angles.json”. Allows to have the actual position stored in the ROM.

7. actualitzarServoAngle(self, servo, angle): Updates the angle “angle” of the servo “servo” in the self.angles dictionary. Also update the file “angles.json” by calling the actualitzarArxiuAngles function. If the servo it “eix” or “manovella” (the main mechanism) it updates the self.x and self.y values by calling the recalcuarPosicio function.

8. actualitzarAnglesEixos(self, pols1,pols2): Updates the current values of the crank
and the axis angles by receiving their pulses. It also updates the “angles.json” file by calling the actualitzarArxiuAngles file, and recalculates the self.x and self.y values by calling the recalcuarPosicio function.


[10] enviarPWM(self, canal, pols): Sends a PWM pulse to the servo on the channel “canal” to move it to the pulse “pols”.

[11] convertirAnglePols(self, angle, servo): Calculates the pulse that corresponds to the angle “angle”. It uses an interpolation between the maximum and minimum values of the angles and pulses stored in “self.dades”.

[12] convertirPolsAngle(self, pols, servo): Calculates the angle of the pulse “pols” that corresponds to the servo “servo”. It uses the same interpolation as on the convertirAnglePols method.

6.3.3. servos.py

- Imports: sleep from time, BaseServos’s class from baseservos, interpolacio’s function from interpolacions and mecanisme.
- Class: Servos(BaseServos).
- Functions in Servo’s Class:

[1] __init__(self, robot=None): Executed every single time an instance of the Servo class is initialized. Inherits the BaseServo’s __init__ method.

[2] moureServo(self, servo, angle, duració=None): Moves the servo “servo” to the angular position “angle”.

[3] mantenirPosicio(self): Sends to all the servos the pulse that corresponds to actual position the position. The actual position is loaded in self.angles dictionary. This method it is very important to make sure the servos remains on the position.

[4] actualitzarSlidersDesplacament(self, servo, angle): Returns a string containing a
coded message to be sent to the interface. It is invoked when the user moves the *crank* or *axis* slider. If the position is invalid, it returns a "0" followed by the *axis* and *crank* positions before the user's move. If the position is valid, it returns "1" followed by the *horizontal* and *vertical* values of the sliders.

[5] actualizarSlidersServos(self, displacement, position): Returns a string containing a coded message to be sent to the interface. It is invoked when the user moves the *vertical* or *horizontal* sliders. If the position it is invalid, it returns a "2" followed by the *vertical* and *horizontal* positions before the user's move. If the position is valid, it returns a "3" followed by the new *axis* and *crank* values.

[6] moureEixos(self,x=None,y=None,duracio=None): Moves head to (x,y). Moves at the same time the axis and crank ("eix" and "manovella" respectively). If only one coordinate is given, the movement will be vertical or horizontal strictly.

[7] resetServos(self): Gets every single servo to the reset position at the same time. Resets the file "angles.json" and the dictionary "self.angles" with the reset position. The reset position it is the dictionary "self.anglesReset" (dictionary in the function definirParametres in Baseservos's class). This method has been very *challenging* because Alejandro's reset method was different. It did not move all the servos to their reset position at the same time. But now, they move step by step and all at the same time. If the position before pressing the reset button is not the reset position, it will return a "4" that will be sent back to the API to update all the sliders to their rest position.


[9] posicioInicialServos(self): Function required every time the WebSocket communication is opened. It gets the servos to the reset position (dictionary "self.anglesReset") directly, without making any interpolation between the current position and the rest one.

6.3.4. mecanisme.py

This file has not been significantly modified during this project.

- Imports: sin, cos, asin, acos, pi and sqrtf rom math, BaseServos's class from baseservos, interpolacio's function from interpolaciones and mecanisme.
- Class: Mecanisme.
• Functions in Mecanisme’s Class:

[1] __init__(self, servos=None): Executed every single time an instance of the Mecanisme class is initialized. It creates the attribute self.servos and calls two functions: self.definirParametres, self.inicialitzarParametres.


[4] avaluarSistema1(self,angle1,angle4): Solves (Eq. 5.1.) and (Eq. 5.2.). Returns the values of Px and Py as a tuple (Px,Py).

[5] avaluarSistema2(self,angle2,angle3): Solves (Eq. 5.3.) and (Eq. 5.4.). Returns the values of P1 and P2 as a tuple (P1,P2).

[6] avaluarRestriccions(self,angle1=None,angle2=None,angle3=None): Solve a pair of equations with the actual values of the angles. Those equations help to prove if it is a correct configuration of angles.

[7] calcularPosicio(self,angle1,angle2): Calculates and returns the position (x,y) (in mm) for the angles configuration input. It first gets angle4 and angle3, with trobarSolucio, and then evaluate equations (Eq. 5.1.) and (Eq. 5.2.) with avaluarSistema1 to get x and y.

[8] calcularAngles(self,x,y): Calculates and returns angle1 and angle2 (ϕ1, ϕ2, possible) (in degrees) for an input position of the point P. The variable possible is a Boolean that means if the returned configuration is valid or not.

[9] resoldreSistema(self,p1,p2,d1,d2): Solves the equations (Eq. 5.5.) and (Eq. 5.6.). Returns the list candidats with possible angle1 and angle4 pairs of candidates.

[10] trobarSolucio(self,candidats,px,servo): Returns the tuple (Beta1, Beta2). This tuple is the correct configuration from among the possible solutions in candidats.
6.3.5.   interpolaciones.py

This file has not been modified during this project.

- **Imports**: `factorial` from `math`.
- **Class**: No class, only functions
- **Functions**:

  1. `interpolacio(passos,duracio,tipus="bezier",c1=0.6,c2=0.92)`: Applies an interpolation of the type “tipus” returning a list with the delay times between the steps “passos” (float with the number of steps). “duracio” stands for the total duration of the movement. If there is no “tipus” input, the interpolation will be a Bezier interpolation. Depending on which interpolation type is needed, the method will call one of the three following functions: `interpolacioSimple`, `interpolacioLineal` or `interpolacioBezier`.

  2. `interpolacioSimple(passos,duracio)`: Returns a list of floats (values of delayed durations). The values will be all the same because it is a simple interpolation.

  3. `interpolacioLineal(passos,duracio,c1,c2)`: Returns a list of floats (values of delay) from a linear interpolation.

  4. `interpolacioBezier(passos,duracio,c1,c2)`: Uses the `bezier` function to get a list of floats (values of delay).

  5. `bezier(u,coefficients)`: Returns the values of the Bezier’s function calculated with the coefficients “coefficients”. It does multiplying the Berstein’s coefficients by the corresponding coefficients.

  6. `berstein(u,i,n)`: Calculates and returns the Berstein’s “i” coefficient, of “n” order, evaluated in the “u” value. Used in Bezier’s interpolation.
6.4. Web-Based API

Controlling the robot using the Omega2 module has been one of the main objectives. Due to the recent release on the market of this module, the range of possibilities for communicating via Wi-Fi to the Omega2 server is very short. This is one of the disadvantages of working with the Omega2. While working with Arduino or Raspberry is very easy because a lot of people use them and internet is full of documentation, working with Omega2 is not that easy.

Only one way to control the robot remotely has been found on a blog [27]. This method is based on a Python framework called Tornado, that establishes a WebSocket server to send messages to the Omega2. In this communication via Wi-Fi the web-based API plays a client role. This API has to be launched in a browser from a computer, smartphone or tablet. This variety makes this communication method a very good system. I must remark that this part has also been a challenge because the knowledge and experience with HTML, CSS, JavaScript and jQuery were null.

6.4.1. API interface

The API (Application Programming Interface) is the user interface. This interface has been created right from the bottom. In this case, the API is a web-based interface, what means that we need a browser to use it. It is from where the robot can be controlled in an easy and intuitive way, without having to code any line.

*Figure 6.7 - Interface [Own source]*.
The interface is mainly made up by 6 linear sliders, one semicircular slider and two buttons.

- **RESET BUTTON**: By pressing it, the robot will move the servos at the same time to get the robot to the reset position progressively.

![Robot on the reset position](image)

*Figure 6.8 - Robot on the reset position [Own source].*

- **AUTOMATIC MOVE BUTTON**: By pressing it, the robot will start a pre-loaded and automatic movement. This action consists of moving three domino's pieces pretending it is playing to domino. I have chosen the domino game to make a demonstration of the robot grabbing and dropping things. This automatic movement is completely random what means that it could be changed. Fig 6.9 shows on the left side the pieces at their initial position. Each piece has a number which indicates the order sequence when each piece is grabbed. On the right side, dominos pieces are placed like if the robot knew how to play.
• **SLIDERS**: On the top-right side there are four lineal sliders. Each one controls independently 4 of the 5 servos. The fifth servo, the one in charged on the rotation and it is controlled by the semicircular slider. Finally, there are two extra sliders: *vertical* and *horizontal*. This pair of sliders move the *axis* and the *crank* servos at the same time. The sliders have on their ends an illustration with a purple arrow showing the movement in case we move the slider to it.

![Figure 6.9 - Dominos pieces position for the automatic movement [Own source].](image)

6.4.2. **API software**

As it has been mentioned before, the aim of the communication is a Python framework called **Tornado**, that establishes a **WebSocket** Server to send messages to the Omega server.

**WebSockets** [28] is a technology that makes possible to open an interactive and bidirectional communication between the user browser and a server. WebSocket communications are not standard HTTP connections but the “handshake” of a new communication is HTTP and then it switches the protocol to a *message-based* communication. So, with the web-based application, we send messages to a server and we can also receive messages back.

In WebSocket communications, clients can be, for example, web browsers, mobile applications or games. In our case, the client is a web browser that runs an HTML API.
**Tornado** [26] is a Python web framework that can scale to tens of thousands of open connections, making it ideal for WebSockets, and other applications that require a long-lived connection to each user.

What we are using from Tornado is a **module** that implements the WebSockets protocol. This module (tornado.WebSockets) is implemented on the tornado.websocket.WebSocketHandler(application, request, **kwargs) class [29]. To create a basic WebSocket handler we just need to subclass this class with an inherit class. Therefore, the aim the file robot.py, is the class: IoTWebsocket(tornado.websocket.WebSocketHandler) that inherits from the tornado.websocket.WebSocketHandler class.

The tornado.websocket.WebSocketHandler class has some method that have been overridden in the inherited class. Those methods are: check_origin, open, on_message and on_close. The implementation of those functions has been full explained in 6.3.1 robot.py.

The Web-based API seen in Fig. 6.7 has been developed with **HTML**, **CSS**, **JavaScript** and **jQuery**. The HTML is the file that has to be opened by the browser, but there are also other files needed. The HTML file has been based the code from [27]. A lot of changes have been made to come up with the final interface. Fig 6.10 explain how the API files are structured.

**Sliders** play a very important part on the interface. Sliders help the user to change the position by dragging them. The sliders are a visual and intuitive solution to the servos. There are two different type: linear [30] and circular [31]. Both are jQuery plugins. By only including some files
(CSS and JS) in our project we can use them. The aim of the sliders is to move the handle and when we released it, a message is sent with the new position. Both sliders needed a tuning work to match their style to the API.

To create the a linear slider we need to create a `div` and then in the scripts parts we must apply the `.slider()` method to be able to personalize the slider. For example, the axis slider:

```html
<body>
...
<div class="eix" onmouseup="moveEix()"></div>
<script>
$('.eix').slider({ min: 0, max: 125, step: 1, value: 110 }).slider("pips", {first: false, last: false, rest: false}).slider("float", {handle: true});
</script>
...
</body>
```

Regarding the communication between the WebSocket Server and the HTML client we need to use the following instances and methods in JavaScript:

```javascript
var ws = new WebSockets("ws://omega-XXXX.local:8888/WebSockets")
ws.send(message)
ws.onmessage = function
```

To establish a communication between the interface and the WebSocket server, we need to create the instance on the first code line. This code line creates a new websocket communication with the server written in the bracket. During the test phase a problem regarding this instance has been found. When I was trying to use the API from a smartphone or tablet, it was impossible to establish the communication but from a computer it was possible. After some test I finally came up with a solution. The problem was that the tablet or smartphone could not understand what the host `omega-XXXX.local:8888` was. By changing this host by the IP address of the Omega2 "192.168.3.1:8888", this issue was already solved.

Methods `.send()` and `.onmessage` must be invoked properly when we want to send messages to the Websocket server or when we want to “listen” to a incoming message from the Websocket server respectively.
Fig. 6.11 is an event diagram between the user, the API browser and the WebSocket server that make easier to understand the JavaScript code above. The method used on this project have been highlighted.

Figure 6.11 - Events diagram between the user, the browser and the WebSocket server [32].

The full code is in ANNEX C.
7. Project Schedule

The current project can be split into seven different activities, which have a determined duration, adding up to a total of around 310 hours, from September 2017 to January 2018. These activities are represented in a Gantt diagram (Fig. 7.1), which shows how this project has been planned. The activities are the following.

1. **Introduction (5 hours)**

This includes the first meetings with the tutor, the definition of the project goals and the explanation of the antecedents and the material provided by the tutor (Omega2+ board, Expansion Dock and the robot)

2. **Omega investigation (33 hours)**

Exhaustive Omega’s documentation analyses. This part has been one of the most important because it is where the potential and benefits of the module were understood.

3. **Robot investigation (15 hours)**

This activity has been to understand the actual status of the robot (components, equations, reading previous reports, etc.)

4. **Coding the main software (100 hours)**

Writing the code has been the most difficult part. It has been a “try and failure” work. It has consisted on writing the driver, understanding the current code and modifying it or implementing new methods.

5. **Developing the API (85 hours)**

This activity is the second one which took the most time. Starting from the bottom and having to learn web developing are the main reasons. Trying to fit the different components into a unique format (colors and shapes) has also been a lot of time.

6. **Final test (7 hours)**
The tests have been done during the entire project. But, at the end some hours have been dedicated to make some user-mode tests to find issues.

7. **Writing the report (65 hours)**

*Figure 7.1 - Gant Chart [Own source]*.
8. Budget

The budget of this project can be split into two main different costs: the cost of the hardware (material) and the personnel cost for developing the application and writing the final report.

Material costs

<table>
<thead>
<tr>
<th>Element</th>
<th>Supplier</th>
<th>Units</th>
<th>Unit price [$/unit]</th>
<th>Price [$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omega2 +</td>
<td>Onion</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Expansion Dock</td>
<td>Onion</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3D part replaced</td>
<td>RepRap</td>
<td>1</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>48</strong></td>
</tr>
</tbody>
</table>

*Table 8-1 - Material costs [Own source].*

It must be remarked that the Expansion Dock can be removed. It has been used to make connections easily but the wires could be welded to the Omega2. To power the board without dock it is needed an external regulator circuit. The price of the components to build this regulator circuit are insignificant compared with the other material prices. This would reduce the material cost by 42%.

Labour costs

Even if all tasks were done individually, they can be unified in two different tasks, with their respective remuneration.
Control of a robotic arm using an Omega2+ module

<table>
<thead>
<tr>
<th>Task</th>
<th>Price [$/hour]</th>
<th>Time [hour]</th>
<th>Costs [$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software developing</td>
<td>33</td>
<td>192</td>
<td>6336</td>
</tr>
<tr>
<td>Writing the final report</td>
<td>20</td>
<td>65</td>
<td>1300</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7636</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 8-2 - Labour costs [Own source].*

Hence, the total cost of the present project is $7684. Applying the current (January 17, 2018) changing value from USA Dollars to Euros, the cost of the project is 6280€.
9. Environmental impact

The environment impact is negligible because the main contribution has been a programming work.

Regarding the material used to build the robot, the PLA “is a biodegradable and bioactive thermoplastic” [33].

It must be remarked that the electronic component send out electromagnetic waves but they do not represent any issue with the environment due to the low intensity.
Conclusions

The purpose of this project was to introduce the Omega2 module in the Department of Electronic Engineering at ETSEIB. This main goal has been accomplished by obtaining a report that can be used as an Omega user guide for further projects.

Furthermore, it has been successfully adapted to a current robot replacing a Raspberry module. Thus, the robot arm’s electronic components price has been reduced considerably. Moreover, a web-based API has been developed to allow the user to control the robot in a very intuitive way.

However, during the project there have been several problems. One of the most important solved problems was that adaptation of a driver to be able to use the current PCA9685 board because there was no driver Omega-compatible already. Other problems came up on the developing and coding phase. Those problems were being solved little by little as they were appearing.

Future recommendations include the dynamic study of robot movement so it could move smoothly. Another recommendation could be the replacement of the front camera so it could be used by the Omega and create an algorithm that could grab automatically.

To sum up, I am satisfied with the work done and with the final result, because it fulfills the initial goals proposed. Moreover, the final product is a low scale robotic arm that can be controlled by a user-friendly interface from the computer, tablet or even from the smartphone.

Figure 0.1 - Controlling the robot from a computer [Own source]
Acknowledgements

I would like to thank Professor Manuel Moreno Eguilaz, for proposing this project but mainly for the continuous support during its realization. He provided all the necessary material. His advice and ideas have been truly helpful, especially at the first steps, as he guided me to know where and how I should start, but also in the last stage, by reading my report drafts and contributing to the work done in this project.
References


