University Carlos III of Madrid

Industrial Electronics and Automation Engineering

Bachelor Thesis

Design, construction and programming of a low cost, Open Source robot for assistive activities

Author
Alvaro Ferrán Cifuentes

Supervisor
Juan González Víctores

ABSTRACT

RESUMEN

Contents

List of Figures

Listings

1 Introduction

2 Socio-economic factors

ageing population, prevision spain, social robots, etc
Android: great market share, cheap
Arduino, raspberry: very cheap 20+35 vs 300 normal pc
3d printer vs injection- / kg vs / kg

3 Regulatory compliance

complies with normativa etc y normativa etc2

4 Scope of the project

This project's ultimate objective is to build a robotic assistant prototype, which may be controlled by a user from their own smartphone and which will stream a video feed from its camera so the user is able to see through the robot. This prototype is built to develop the technology that can be later used in a full-scale robot capable of assisting people with disabilities.

In order to achieve this result, the project is divided into the following objectives:

I. Study of different robot configurations to find the optimal choice for domestic assistive activities.

II. Design of the aforementioned robot.

III. Creation of the designed parts with a 3D printer.

IV. Assembly of the robot's body and installation of the electronic components.

V. Program the microcontroller to understand the user's commands and control the actuators accordingly.

VI. Program the computer to:

   i. set up a wifi network to communicate with the user
   ii. stream video from the attached camera
   iii. send the user's commands to the microcontroller.
   iv. do all of the previous automatically when the system boots.

VII. Program the Android application the user will use to control the robot and receive the video feed it streams.

5 Project phases

[include Gantt diagram]

6 Project components

This section will explain the different tools and components used for the creation of the proyect.

6.1 3D Printer

3D printers are Computer Numerical Control (CNC) machines that are capable of transforming virtual 3D models created with a Computer Aided Design (CAD) software into real-world objects.

Created in 1984 by Chuck Hull of 3D Systems Corp this technology was little-known to the general public and was mainly used in industries for short runs of difficult pieces.
In 2005 Dr. Adrian Bowyer, from the University of Bath, UK, started the RepRap project. Its goal was "to produce a pure self-replicating device not for its own sake, but rather to put in the hands of individuals anywhere on the planet, for a minimal outlay of capital, a desktop manufacturing system that would enable the individual to manufacture many of the artifacts used in everyday life"

Today a vast range of 3D printers co-exist, varying in size, price and materials used.

In this theses a RepRap Prusa Air 2 (Figure 1) model is used. It is of a "fused filament fabrication additive manufacturing" type. This type of printers extrude mainly ABS or PLA plastics, and deposit new liquified material over ther previous layer, now solid, effectively building parts from the bottom up layer by layer.

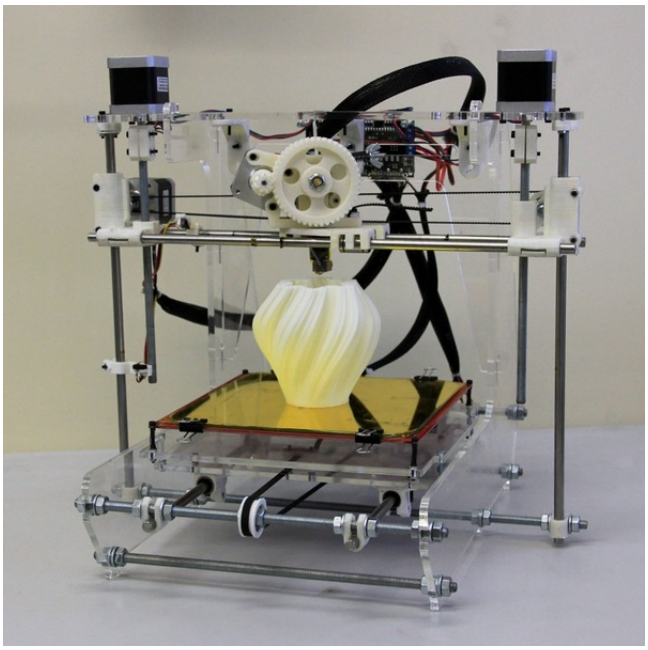

Figure 1: Prusa Air 2 3D printer

6.2 Software

This type of 3D printers work by turning 3D models into plastic parts. These models are first modelled in a CAD program and then processed with a *slicing* software to divide the model into layers of G-code, which is the standard language interpreted by CNCs. This is then introduced into the printer to build the part.

6.2.1 3D Modelling

In this project Sketchup has been used to create the printed parts. Owned by the company Trimble Navigation it is a WYSIWYG (What You See Is What You Get) modelling editor with a large online warehouse of parts available for download. Figure 2 shows the program's user interface.

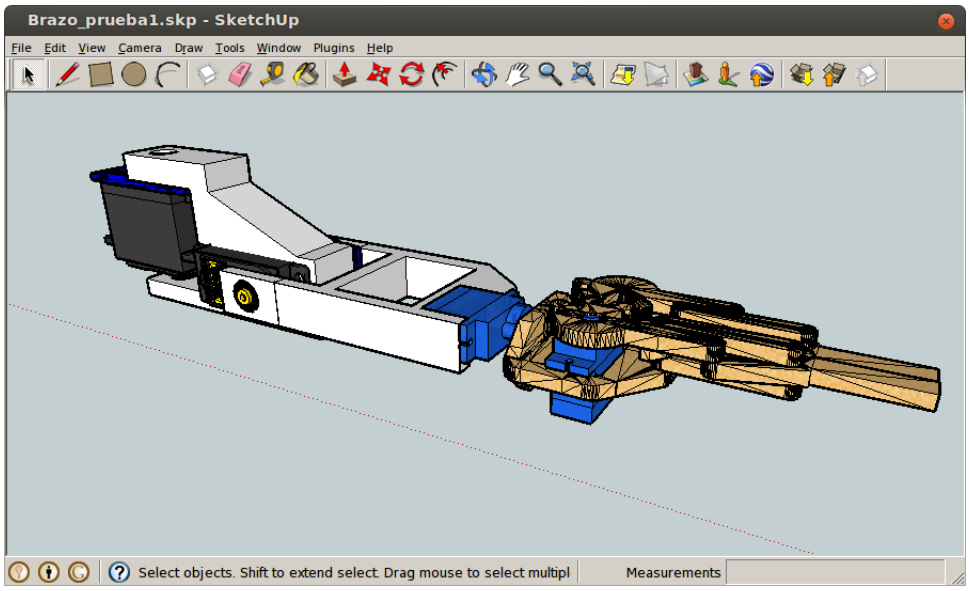

Figure 2: SketchUp software

In order to make it compatible with the slicing sofware, Sketchup's propietary format, *SKP*, has to be converted to the standard *STL*. In order to accomplish this the *Su2stl.rb* plugin is installed. A new *Plugins* menu appears in Sketchup which contains the Import/Export options, where the desired output format and model units are specified.

6.2.2 G-Code Generator

Once the model is converted to *STL* it then has to be sliced. Since 3D printers work by building layer upon layer of plastic, the model has to be transformed into the same format. The G-code generator converts the CAD model into layers of CNC instructions. There are three main slicing programs, each with their own benefits:

- Skeinforge:
  The first slicing program used in homemade 3D printers. It is by far the most complete of the three. It allows the user to control each and every imaginable setting of the printer, from the axis' speeds to the retraction distance of the plastic into the extruder

while moving. However, because of this it has a very steep learning curve which makes it unsuitable for the average consumer.

- Slic3r:
  Slic3r was created as an user-friendly software, which only gives the final user a choice in the basic settings, such as printing speeds, filament widths or part infills. As a result it is an easier program to slice parts with a sufficient level of customization. It has nonetheless problems converting models with imperfections or broken shapes.

- Cura
  Finally, Cura is also designed with user-friendliness in mind. This slicer is more robust than Slic3r, in that it will accept models with imperfections, and will try to correct them. It also features a box simulating the print area in which the model can be moved around, turned or scaled before printing. This last feature is specially useful if minor changes need to be made, without returning to the CAD software.

To print the needed parts Cura was chosen because of its simplicity and usefulness in rearranging the objects without having to modify the original CAD files.

6.3 Li-Ion Battery

The whole system is powered by a lithium ion 12V 6800mAh battery as seen in Figure 3. This was chosen because of the high voltage and durability it delivers over regular AA batteries.

Figure 3: Li-ion 12V 6800mAh battery with charger

6.4 Voltage level converters

Different electronics demand different power levels, ranging from 3.3V up to 12V in this case. In order to provide suitable voltages to each part, different voltage level converters are used.

6.4.1 DC-DC Step-Down Converter

Voltage level converters are used to adapt a source's voltage to that required by the load. In this thesis a DC-DC converter is used to decrease the 12V given by the battery to the 5V

required by the logic components as well as the servomotors.

The simplest method would be to use a linear regulator such as a 7805, which is a cheap, single-component solution. However, this is greatly inefficient solution, since a great part of the power is dissipated as heat. For instance, if a 7805 were to be used in this project, about $\frac{Power_{in}-Power_{out}}{Power_{in}} = \frac{12 \cdot I - 5 \cdot I}{12 \cdot I} = \frac{12-5}{12} = 58.33\%$ of the power is wasted.

A much more efficient solution is to use a switching regulator such as a Buck converter, which has an efficiency level of around 95% [1]. Buck converters work by switching rapidly between "On" and "Off" states, which sets the output voltage in function of the duty cycle $d = \frac{time_{on}}{time_{on}+time_{off}}$. The converter (Figure 4) used includes a potentiometer to set the output level by selecting the duty cycle.

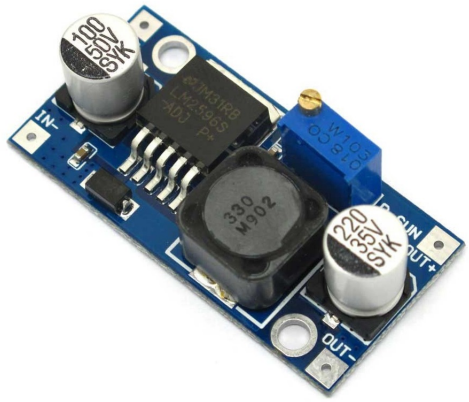

Figure 4: LM2596S step-down converter

The converter's electrical specifications are:

- Adjustable input voltage: 3.2 - 40V
- Adjustable output voltage: 1.25 - 35V ($V_{in} > V_{out} + 1.5V$)
- Max. output current: 3A

6.4.2 Bidirectional logic level converter

In order to enable serial communication between the Arduino and the Raspberry Pi another voltage level converter must be introduced, since the former operates at a 5V level while the latter does so at a 3.3V level.

In this case a switching regulator like the previous one will not work because the communications are much faster than the regulator's switching speed. Therefore a bidirectinal, low power converter can be built out of transistors.

Figure 5 shows a simple converter model. Analyzing the circuit from the low side:

- If a logic one is emitted, the transistor source pin is grounded and it switches on, pulling down the high side to zero.

- If a logic zero is sent, the transistor is tied high and so is off, leaving the high pin connected to the pull-up resistor and thus seeing a one.

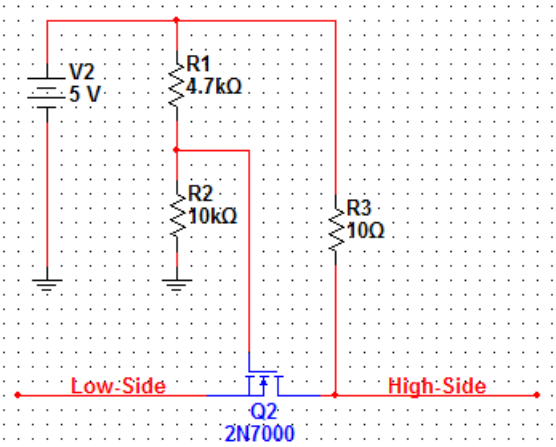

Figure 5: Bidirectional voltage converter

This setup works for one line, two identical circuits are needed in order to provide for serial communication. For this project a commercial board (Figure 6) is used to reduce the total size of the converter by using SMD components.

Figure 6: JY-MCU 5V-3V converter

This board is used with UART communication, but is equally adequate for I^2C, SPI or one-wire communication.

6.5 Motors

Three different types of actuators are present in the robot, which are either continuous motors or servomotors.

6.5.1 DC motor

DC motors are simple actuators that start spinning whenever there is a voltage applied between their wires. Their speed is directly dependent of the voltage level applied, and the turning sense on the polarity of the conection.

This robot uses two GA25Y370-362 dc motors (Figure 7), which turn at a 10rpm with a torque of 5Nm when connected to a 12V source.

Figure 7: GA25Y370-362 motor

Since the motor only has two wires, a controller is needed to interface it to the arduino. This type of controller is called an H-bridge because of its circuit topology (Figure 8). It consists of a series of transistors that can be opened or closed to reverse the polarity of the motor, as well as to apply a PWM modulation to control its speed.

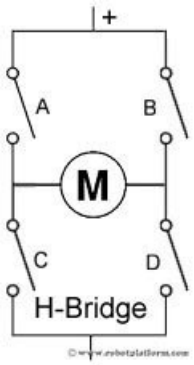

Figure 8: H-bridge circuit

In this case a L293D chip with two internal H-bridge circuits is used to control both motors.

6.5.2 Servomotors

On the other hand, servomotors are much more complex than continuous motors. These consist of a regular dc motor, a set of gears, an internal potentiometer and a control circuit. They also do not revolve continuously but rather oscillate in a range of approximately 180°.

In order to control the servo the user gives the desired position through the input cable and the circuit board matches it to a resistance value. The motor then starts turning which does the same to the potentiometer, and stops when the latter reads the desired resistance value.

The robot uses three GOTECK GS-551MG (Figure 9) servos per arm, which are used for the more demanding movements, while it employs two aditional TowerPro SG90 (Figure 10) servos for the less challenging actions of wrist rotation and gripper manipulation.

Figure 9: GOTECK GS-551MG servo

The GOTECK's technical specifications are:

- Operating voltage: 4.8 V
- Maximum torque: 1.3 Nm
- Speeed: 0.20sec/60deg

Figure 10: TowerPro SG90 servo

The TowerPro's technical specifications are:

- Operating voltage: 4.8 V
- Maximum torque: 0.18 Nm
- Speed: 0.10sec/60deg

6.6 Arduino

Arduino is a family of low-cost electronic boards designed to be easily programmable. From the official Arduino website, "Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. "

Arduino is programmed using its own language, which is merely a set of C/C++ functions compiled with *avr-g++*. They can nonetheless be programmed in pure C or C++ in an external IDE and have code uploaded to it as any other AVR board.

In this proyect an Arduino Nano v3 with an ATmega 328 microcontroller has been chosen mainly due to its processing power and reduced size.

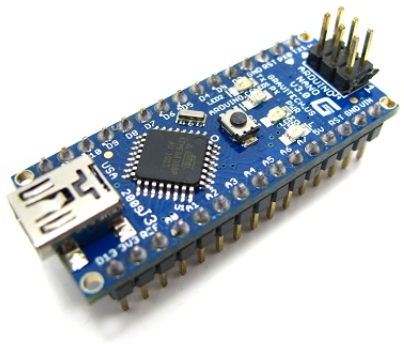

Figure 11: Arduino Nano v3

The official Arduino Nano V3 specifications are:

- **Microcontroller:** Atmel ATmega168 or ATmega 328
- **Operating Voltage (logic level):** 5V
- **Input Voltage (recommended):** 7-12V
- **Input Voltage (limits):** 6-20V
- **Digital I/O Pins:** 14 (of which 6 provide PWM output)
- **Analog Input Pins:** 8
- **DC Current per I/O Pin:** 40mA
- **Flash Memory:** 16 KB (ATmega168) or 32 KB (ATmega328), of which 2 KB used by bootloader
- **SRAM:** 1 KB (ATmega168) or 2 KB (ATmega328)
- **EEPROM:** 512 bytes (ATmega168) or 1 KB (ATmega328)
- **Clock Speed:** 16MHz
- **Dimensions:** 0.73" x 1.70"
- **Communications:** UART, SPI and I^2C buses

6.7 Raspberry Pi

From the official website of the homonymous foundation, the Raspberry Pi is a "credit-card sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects."

Figure 12: Raspberry Pi model B

Available in two models, A and B, the Raspberry has a Broadcom BCM2835 System On a Chip, which includes an ARM1176JZF-S 700MHz processor and a VideoCore IV GPU. It includes as well a 256Mb RAM, upgraded to 512Mb in model B.

The Pi features:

- HDMI, composite and raw DSI video outputs
- 3.5mm audio jack
- SD card socket
- Low-level peripheral connections including:
  - 8 General Purpose Input Output (GPIO) pins
  - Universal Asynchronous Receiver Transmitter (UART) bus
  - Inter-Integrated Circuit (I^2C) bus
  - 2 Serial Peripheral Interface (SPI) buses
  - Power pins: 3.3V, 5V and GND
- Ethernet socket
- USB hub (1 socket in model A, 2 in model B)

The main storage unit is the SD card, and that is where the OS is flashed, normally a Linux distribution. The most popular is Raspbian, an adapted version of Debian Wheezy, although other Linux distros or even other OS like Android or XBMC can be used.

(a) WiFi USB dongle (b) PlayStation 2 EyeToy (c) LCD screen 16x2

Figure 13: Raspberry Pi peripherals

In this proyect a Raspberry Pi model B running Raspbian manages the software side of the robot. It has an Ralink Technologies RT5730 WiFi USB dongle , a PlayStation 2 EyeToy USB camera and a 16x2 character LCD screen connected in order to create a WIFI Access Point, stream video to the user and signal its status respectively.

6.8 Android Phone

Android is one of the most popular operating systems for smartphones. Created in 2003 by the eponymous company and adquired by Google Inc. two years later, its open-source nature has incentivised manufacturers to include it in their products, expanding its reach until it has dominated the market share, as seen in Figure 14.

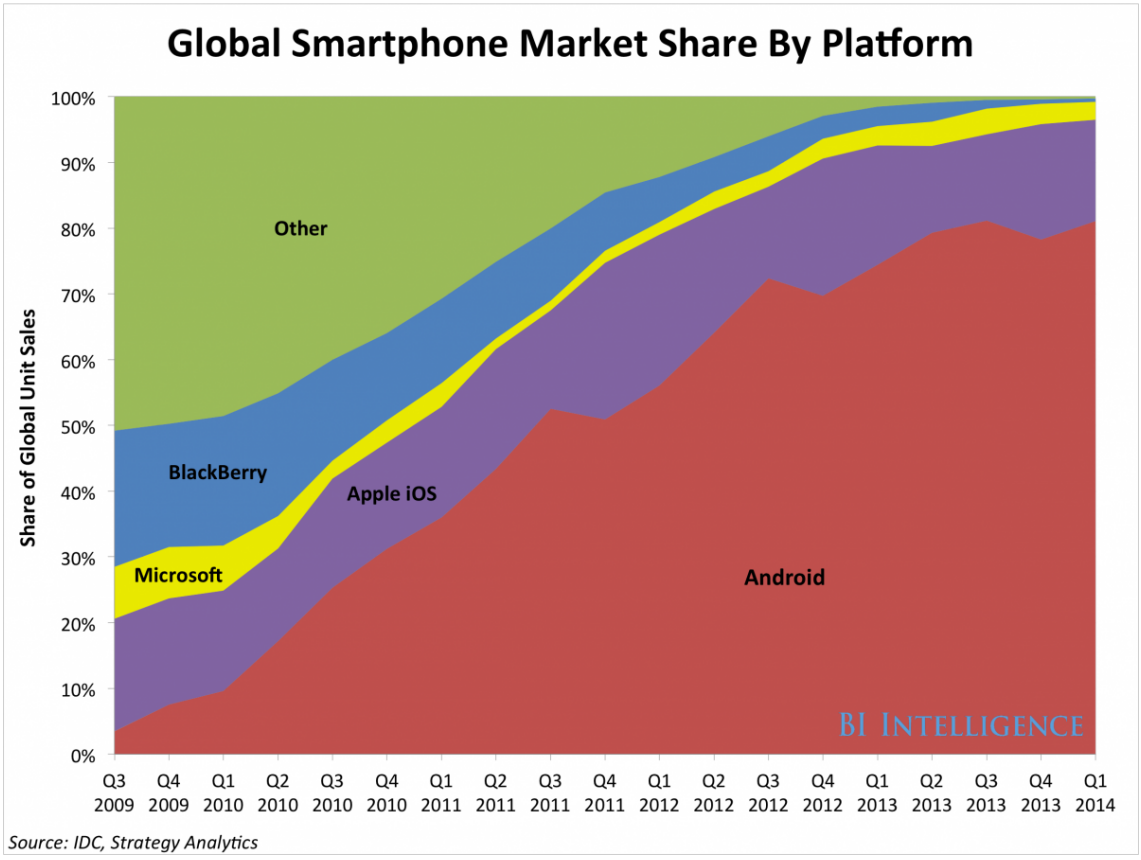

Figure 14: Smartphone market share

Android has been chosen for the following reasons:

- **Market share:** Being the operating system used by the highest number of people means that the robot will be useful to the largest amount of users possible.
- **Price ranges:** Because most mobile phone manufacturers nowadays include Android each user will be able to enjoy the product whithout having to pay for a high-end phone.
- **App development:** Finally, Android apps such as Bot Control can be programmed from any computer regardless of the OS, and can be uploaded to the user's phone directly. This means users are able to customize the app with no extra costs, such as having to use a certain OS and paying a developper's fee.

In this project a Haipai Noble H686 (Figure 15) is used. Some of its more relevant specifications are:

- **CPU:** Quad-Core Mediatek MTK6589 1.2GHz
- **RAM:** 1 GB
- **OS:** Android 4.2 Jelly Bean
- **Screen Size:** 6.0 inches

Figure 15: Haipai Noble H868

7 Hardware assembly

7.1 Assembly

7.1.1 Upper body

denavit hartenberg for arms

7.1.2 Lower body

election of wheels vs legs

7.1.3 Assembly

This section will cover how to put together all the individual pieces that compose the robot in a step-by-step guide.

The first part to be assembled is the one with the greatest number of pieces, the gripper. This was designed by XXXXXXXXXXXX, and is used because of its parallel way of closing, which guarantees a good grip at any degree of aperture. In order to build it, the corresponding lettered holes simply have to be connected, as shown in Figure 16. These must be fastened together with M3x25mm screws, except for hole "A" and the wrist servo, which are fastened with the bolts included with the servos.

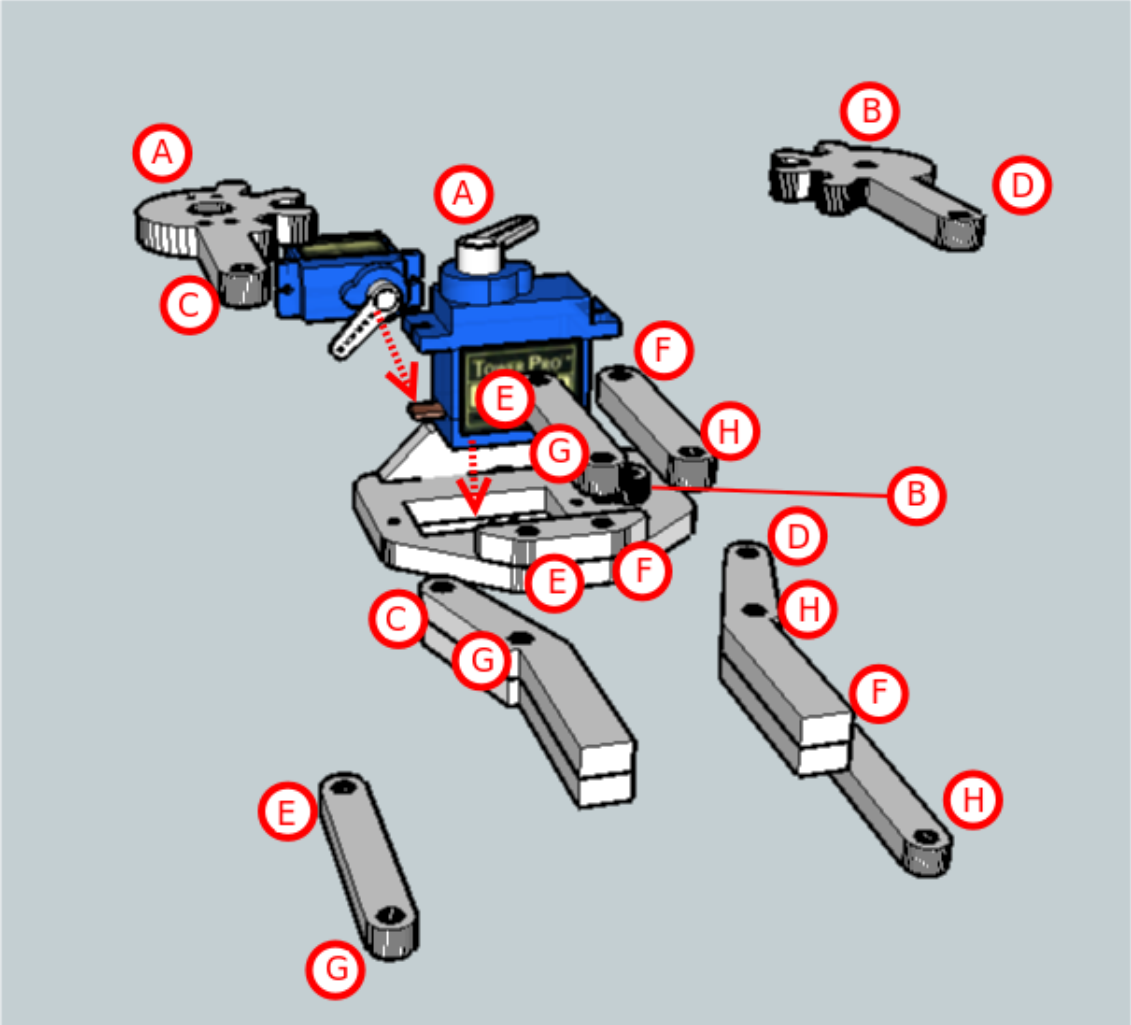

Figure 16: Assembly of the gripper

The end result can be seen in Figure 17, which can also be used for reference during assembly.

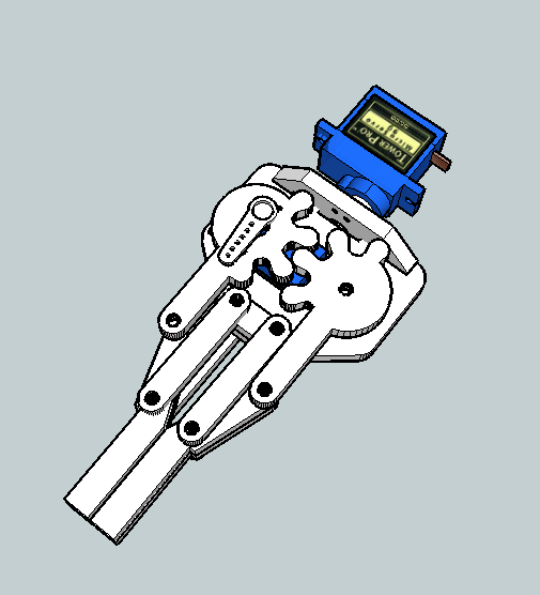

Figure 17: Detail of the gripper assembled

The next step is to attach the forearm to the gripper. This is done by snapping the wrist servo into its socket and securing it in place with screws.

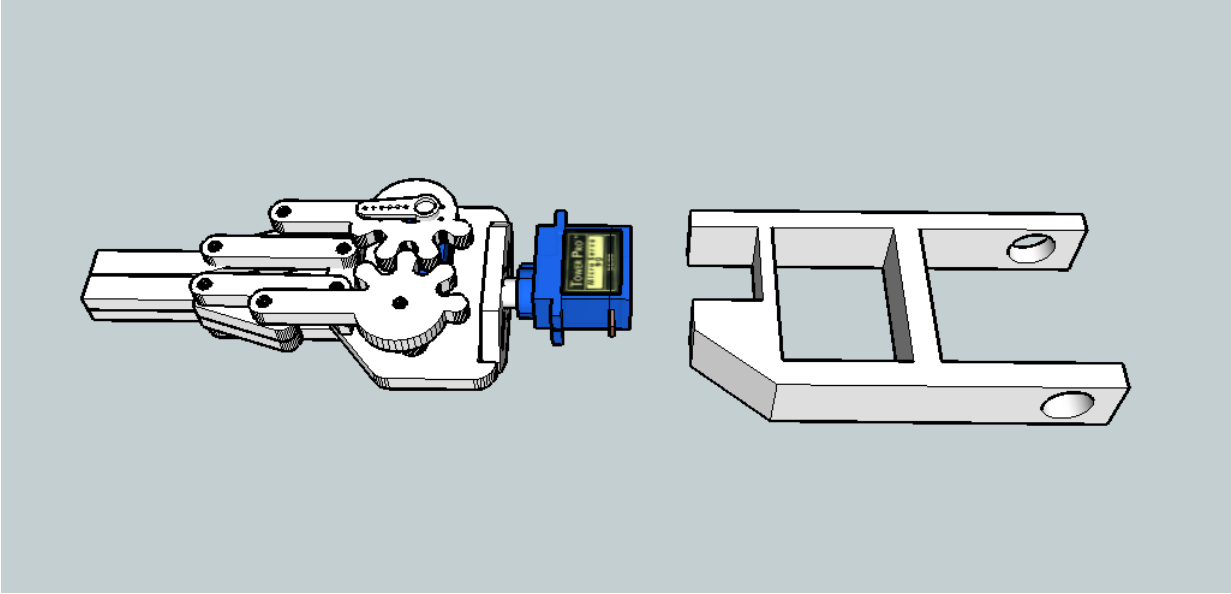

Figure 18: Connection of the gripper to the forearm

Once the gripper and forearm are attached, the elbow needs to be assembled (Figure 19). This is done by introducing one of the Goteck servos through the hole and screwing a shafted lid designed by (obijuan) .

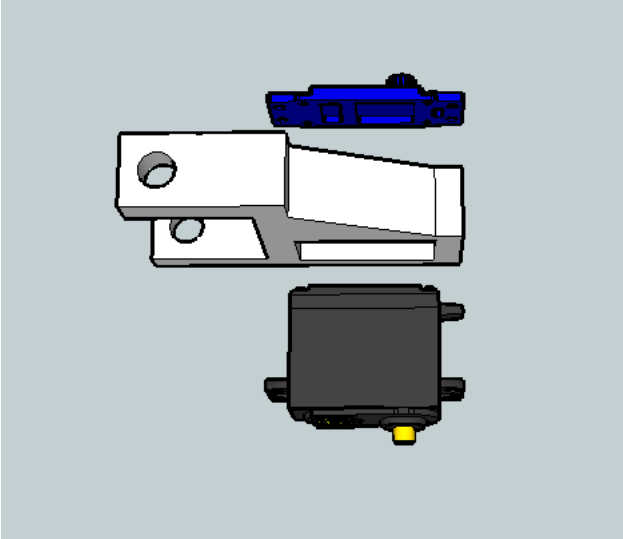

Figure 19: Detail of elbow assembly

Figure 20 shows how the elbow is snapped into the rear of the forearm, by bending slightly the latter's prongs.

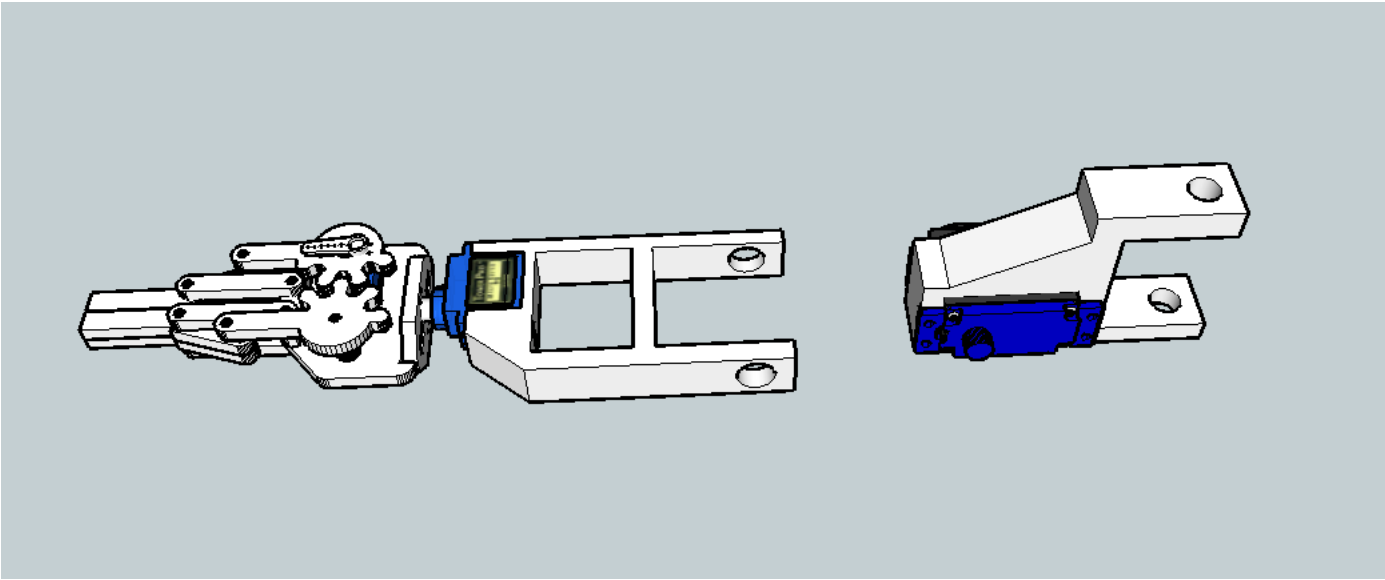

Figure 20: Linkage of the elbow to the forearm

The upper arm module is fitted with a Futaba servo in the same fashion as the elbow (Figure 21) and then fitted into the elbow piece in the same manner as the previous unit (Figure 22).

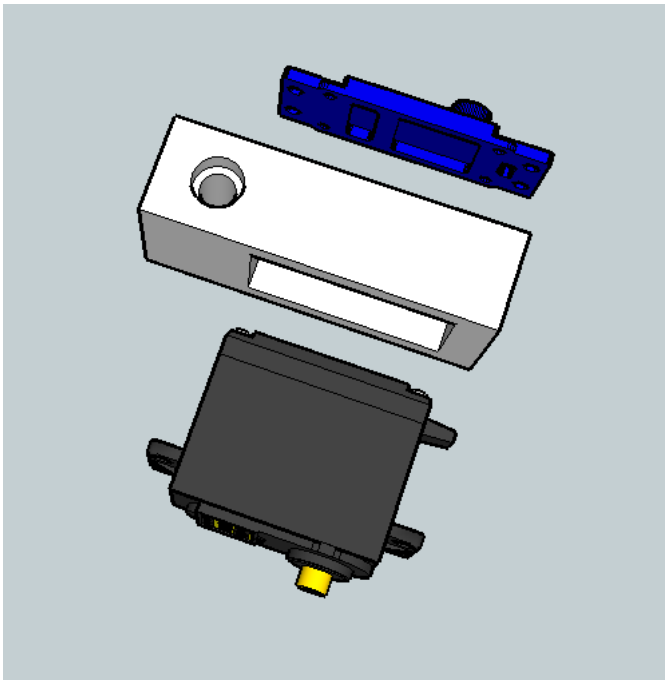

Figure 21: Detail of upper arm assembly

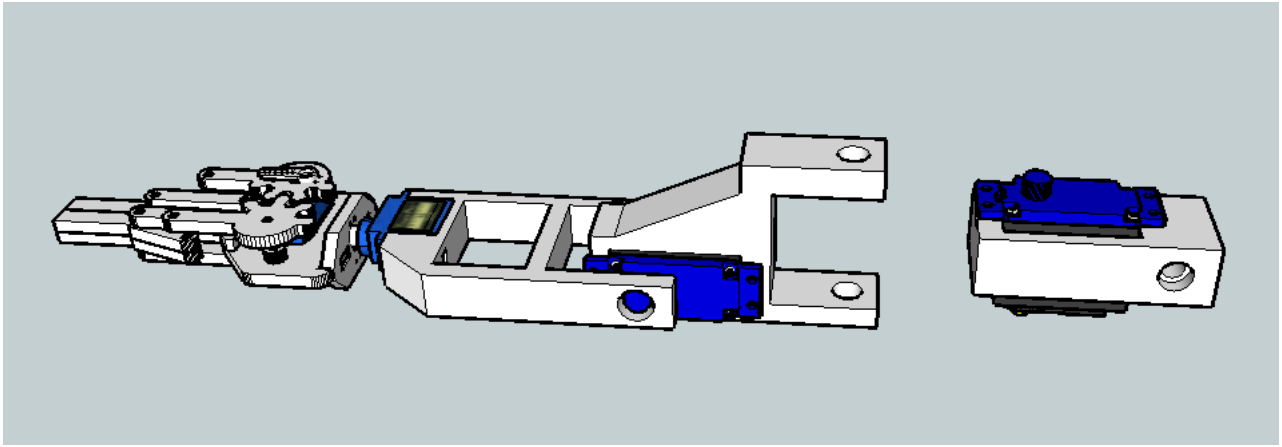

Figure 22: Coupling of the upper arm to the elbow

Next, two 623ZZ bearings are placed each into the upper arm (Figure 23) and shoulder (Figure 24) modules. The latter also has a Goteck servo inserted in place.

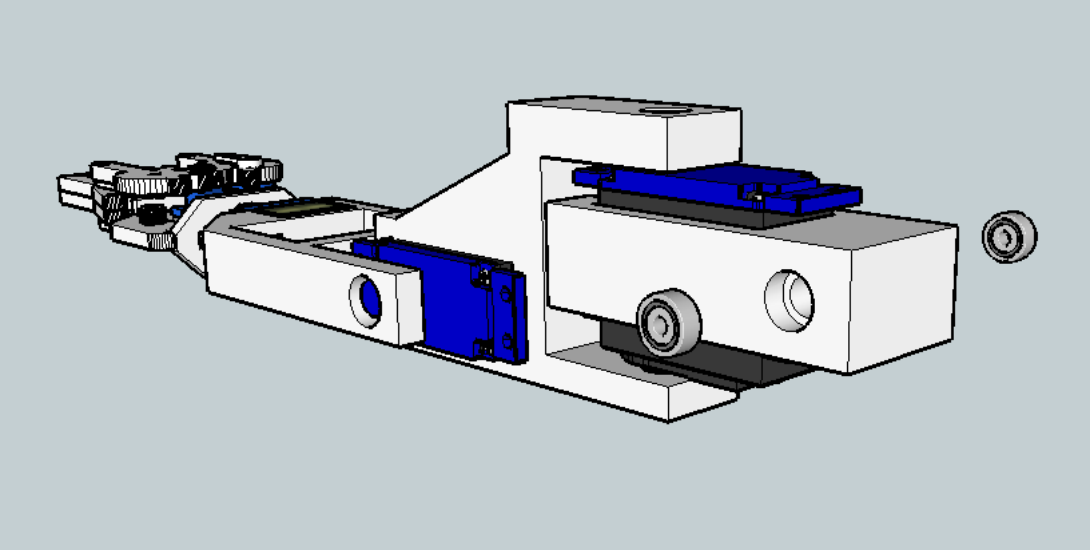

Figure 23: Detail of bearings insertion into the upper arm

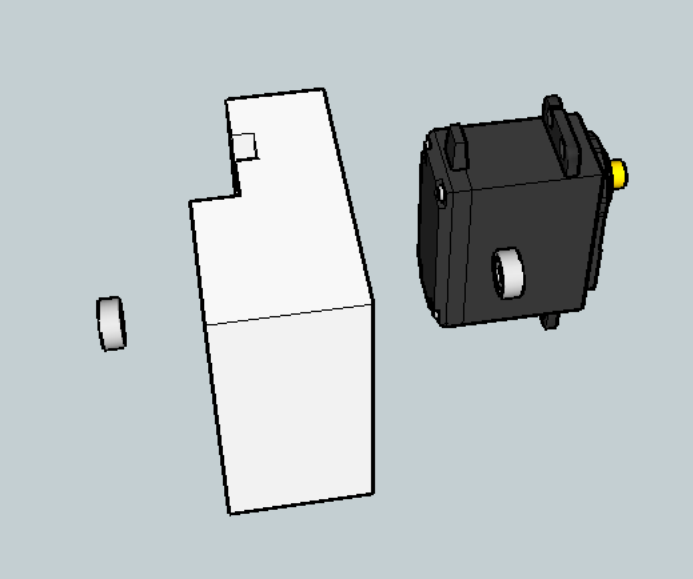

Figure 24: Detail of shoulder assembly

The completed arm is linked to the shoulder part through a M3x120mm threaded rod and secured in place with four M3 nuts, which will avoid lateral displacement (Figure 25). This is done to discharge the servomotor from the arm's weight, so it only needs to provide torque for turning, but not have to support the load.

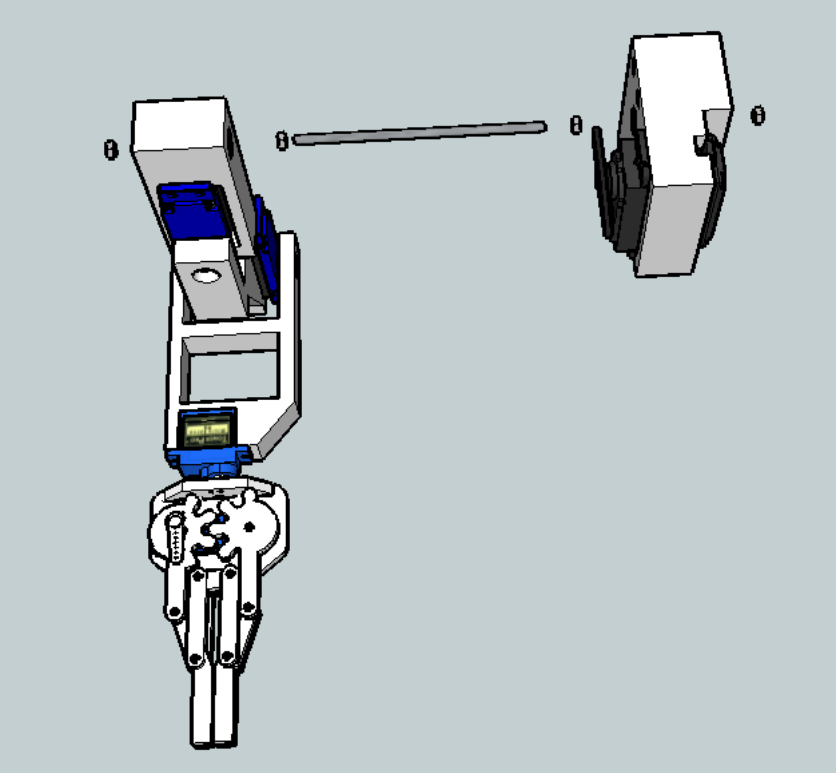

Figure 25: Coupling between arm and shoulder

Figure 26 shows the mechanical linkage between the servo and the upper arm module, which is able to rotate freely at both ends, providing traction to move the arm around the shoulder axis.

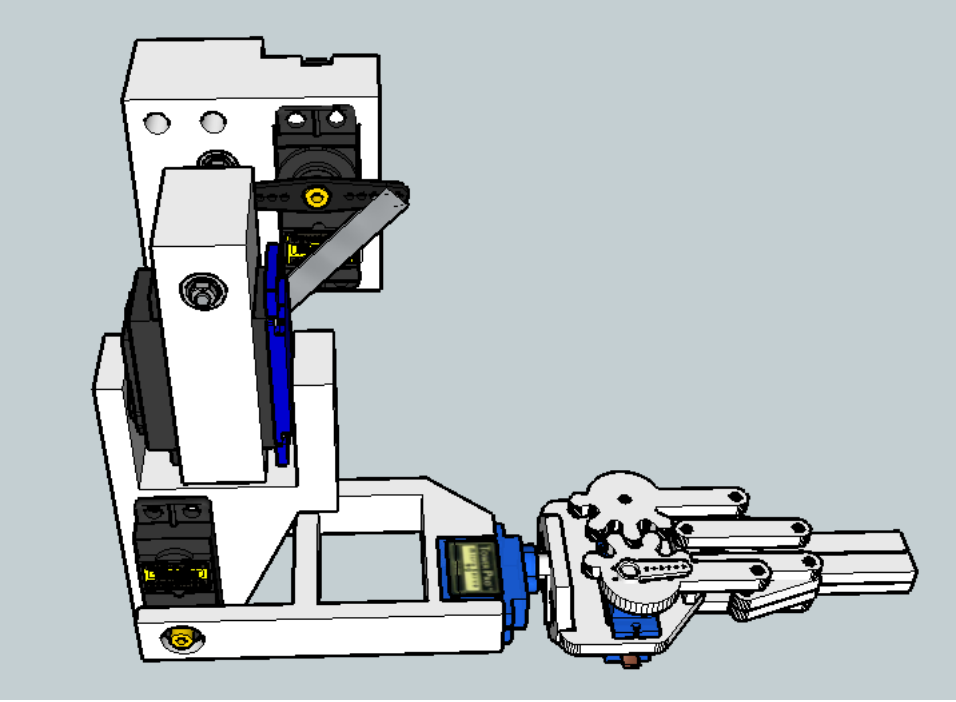

Figure 26: Detail of linkage between the shoulder servo and the upper arm

Figures 37 and 28 show how the Raspberry Pi and the webcam are secured to the robot's head with two stripes of 20x80mm velcro for the former and one stripe of 20x40mm velcro for the latter.

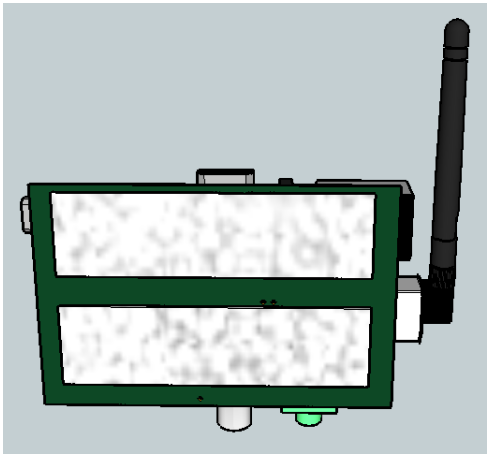

Figure 27: Detail of velcro stripes used to secure the Raspberry Pi

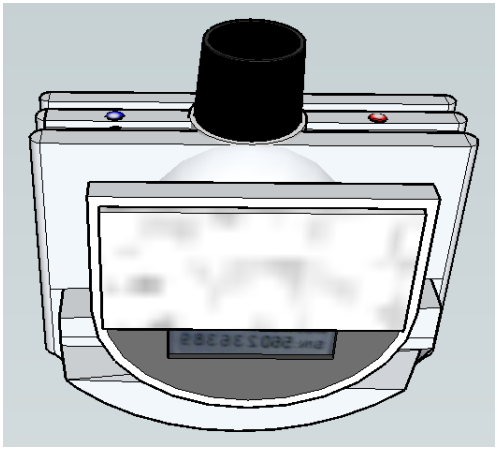

Figure 28: Detail of velcro stripe used to secure the camera

Similarly, both the Raspberry and the camera are attached to the head module through yet another set of two 20x80mm and one 20x40mm stripes of velcro, as seen in Figure 29.

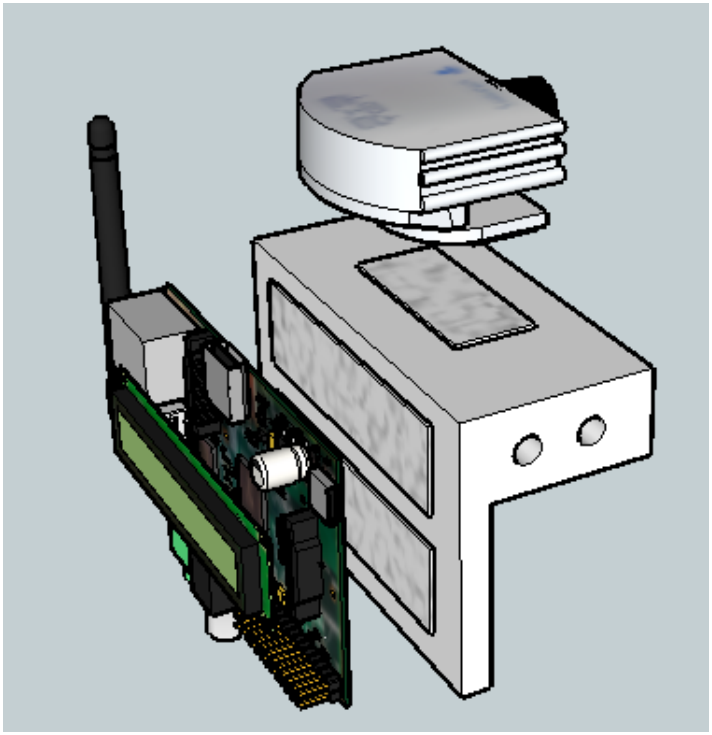

Figure 29: Coupling of the electronics to the robot's head

The procedure is repeated for the other arm, and both of these together with the head are fastened via two M5x250mm threaded rods and a set of six M5 nuts per rod to hold the different modules in position relative to each other (Figure 31).

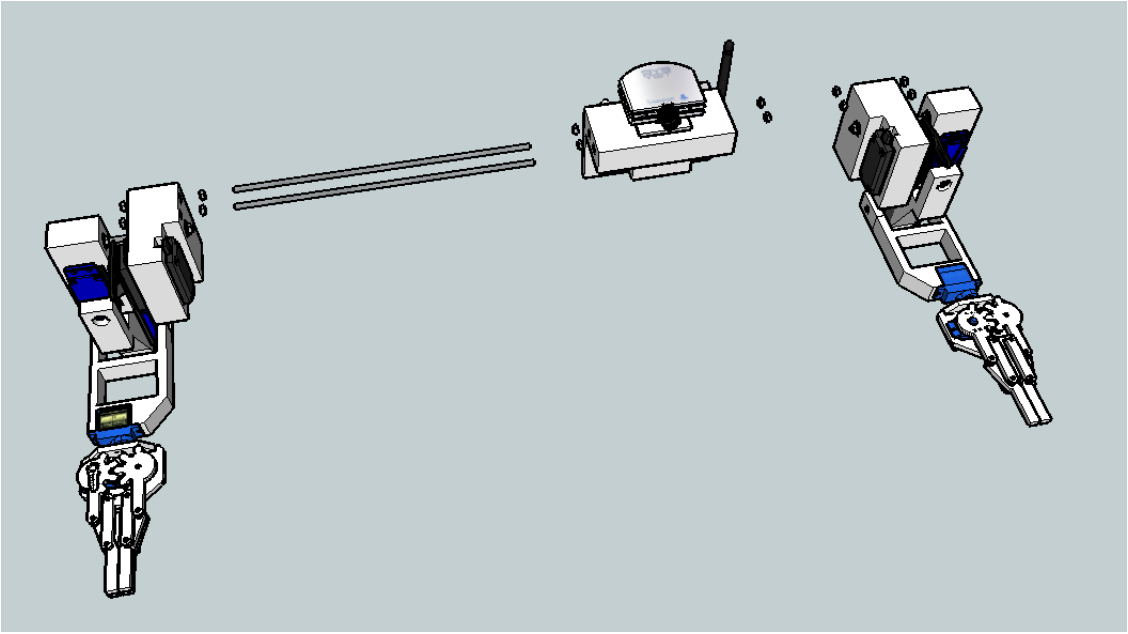

Figure 30: Assembly of the upper body

As shown in Figure 31, the upper body is kept in positon by three pairs of rods, which prevent the body from tilting on either side nor falling forwards due to the arm's torque. The lateral rods are M5x220mm while the central ones measure 250mm while also being M5.

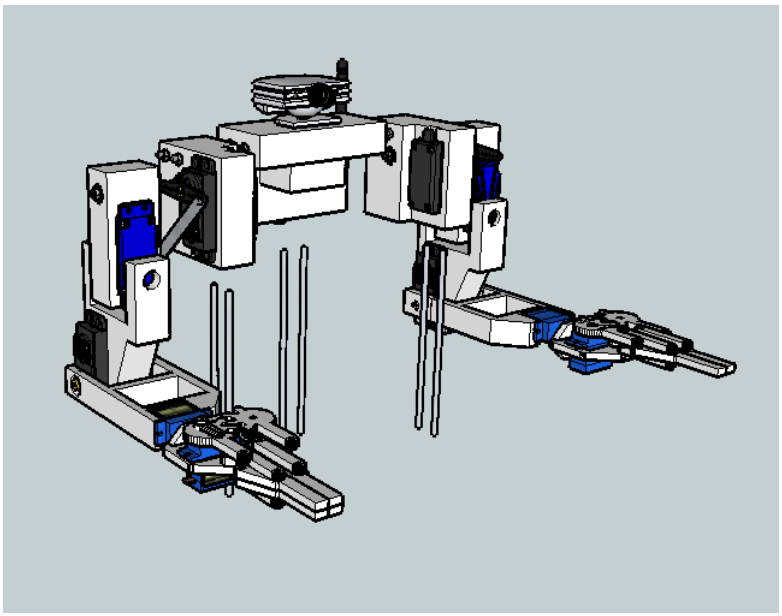

Figure 31: Detail of support rods insertion

The robot base consists of a 200x200x30mm wooden plank, chosen due to its resistance and as a faster and cheaper solution for a plane than a 3D printed part. The two motors are snapped into the wheels and fastened to the plank with zip ties while the caster wheel is screwed with four self-tapping screws (Figure 32).

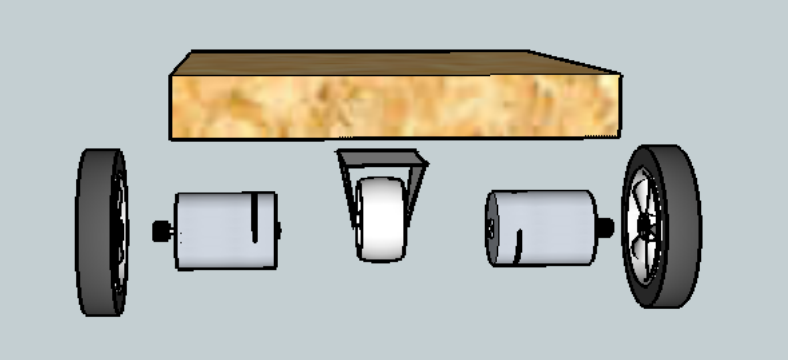

Figure 32: Detail of base assebly

The upper body and the base are then joined together as illustrated by Figure 33, by inserting M5 nuts at either side of each of the base throughout the rods.

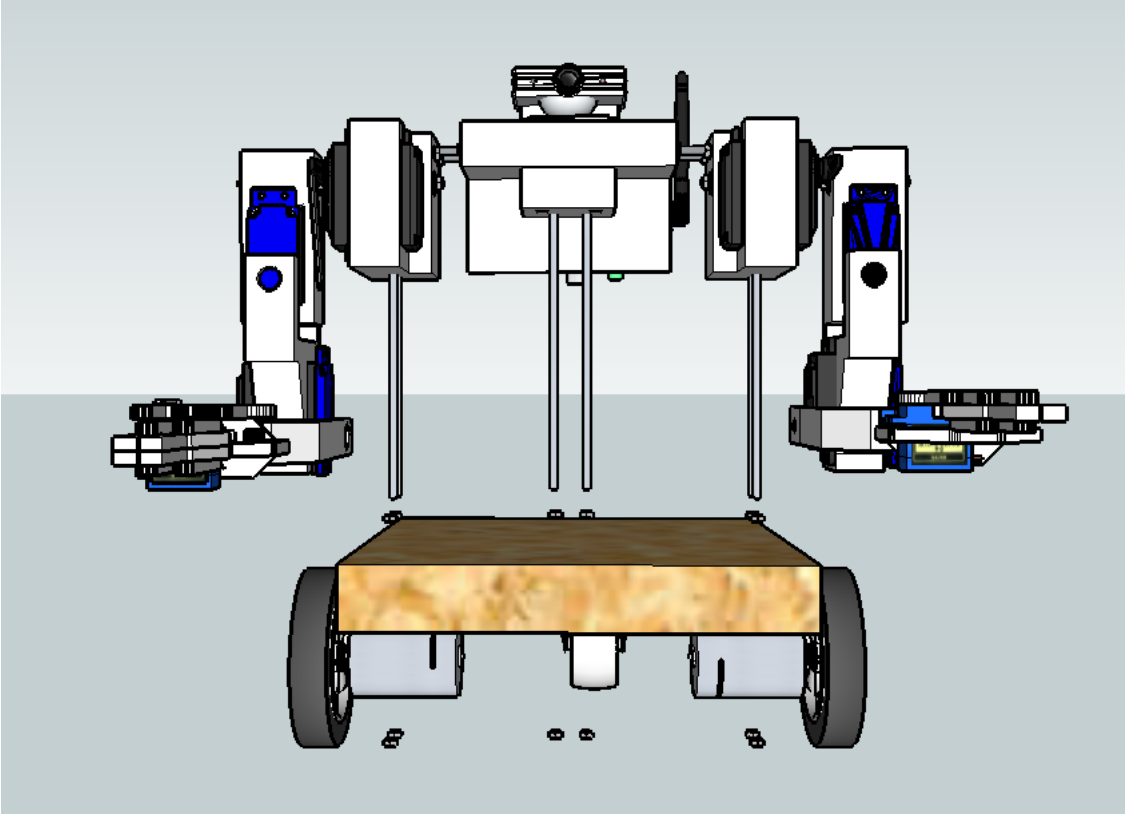

Figure 33: Connection of upper body to base

Figure 34 exhibits the disposition of two more 20x180mm velcro stripes intended to secure in place both the battery and the circuit board containing the Arduino, level converters and motor driver.

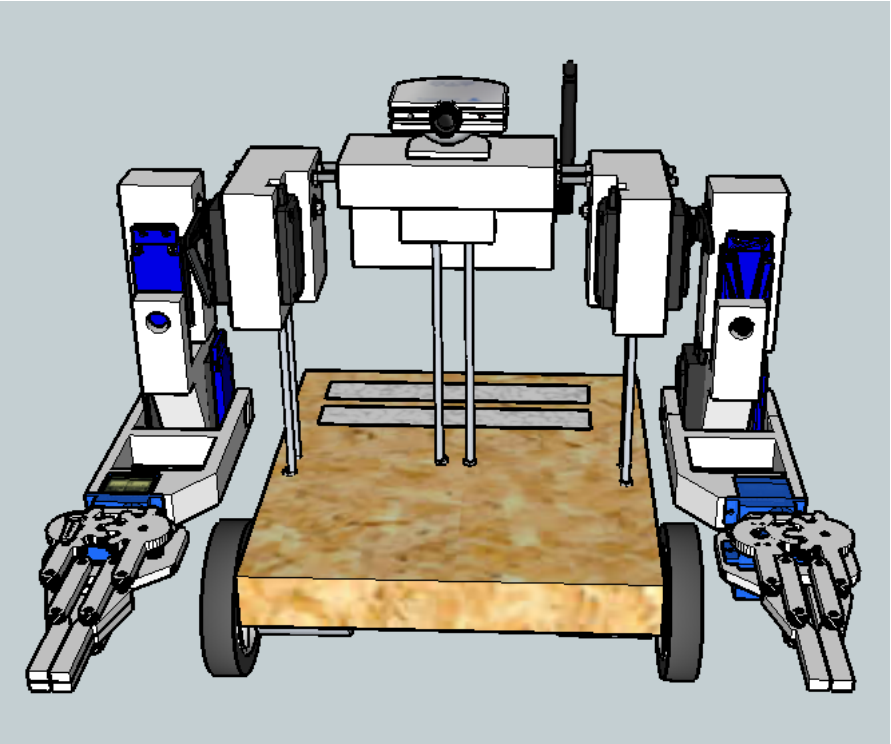

Figure 34: Detail of velcro stripes in the base

Figure 35 presents the velcro stripes at the bottom of the battery and circuit board. These are 20x100mm in dimension and are to be connected to the ones on the base of the bot.

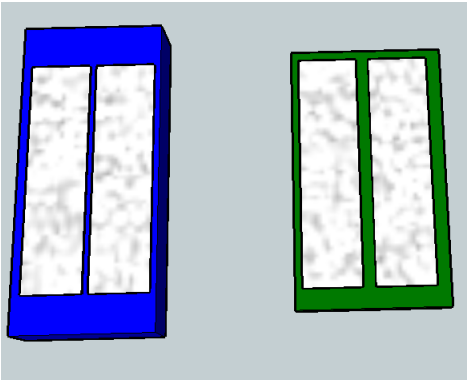

Figure 35: Detail of velcro stripes in the battery(L) and circuit board(R)

Finally, Figure 36 shows the robot fully assembled and in resting position.

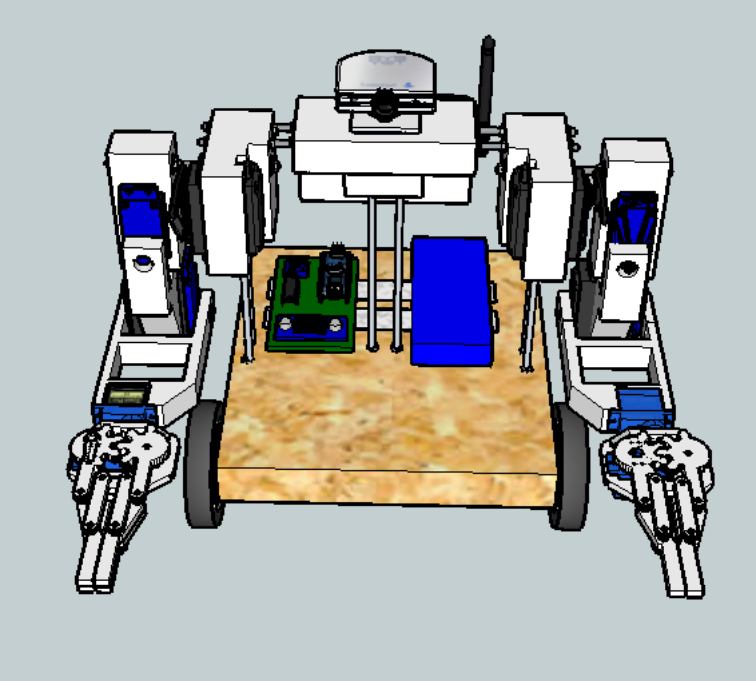

Figure 36: Assembled robot

Figure 37 lists all of the robot's actuators for later identification, classifying them into servomotors (S) or motors (M) and left(L) and right (R) sides.

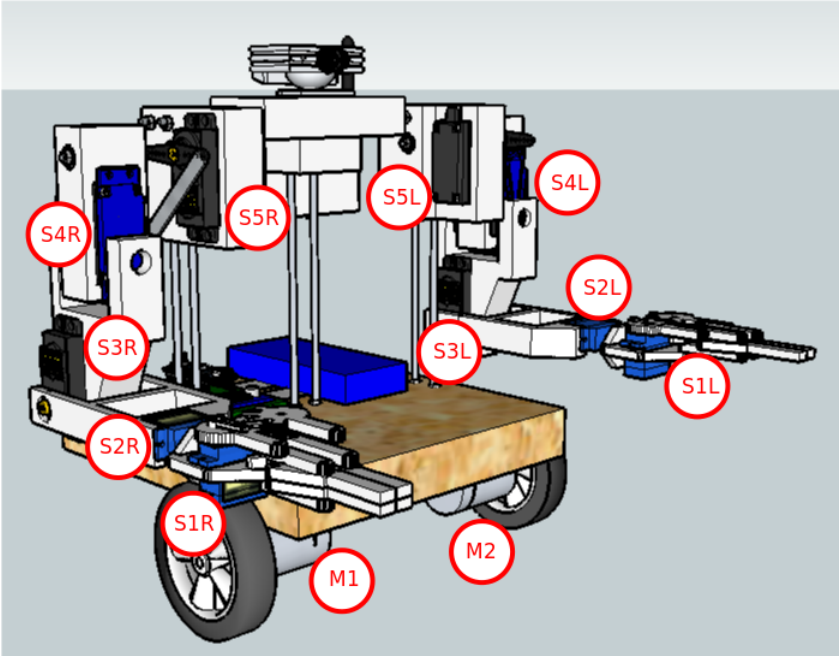

Figure 37: Detail of the robot's actuators

7.2 Connections

This section will present how the different elements composing the robot are connected, first from the electrical point of view, then from a means of communication angle and finally from the different programs' interactions perspective.

7.2.1 Electrical connections

Figure 38 shows how the different electric and electronic components are interconnected. As it can be seen, different voltage levels co-exist within the robot, so regulators are placed to ensure the components function correctly.

The DC motors need the highest voltage to work, and so are connected to the battery, which provides them with the 12V they need. However they have to be controlled by the Arduino, hence the need for a driver that will turn on and off the 12V rails from 5V signals.

The rest of the components operate at 5V, which is why the step-down converter is used to convert the battery's 12V output into the desired level. The Raspberry Pi, Arduino and servomotors are connected to this rail.

Finally, the Raspberry communicates with the Arduino through the former's UART pins, which operate at a 3.3V level and can be damaged by the latter's 5V level pins. To avoid this a logic level shifter is introduced, which ensures data transmission without compromising the hardware's integrity.

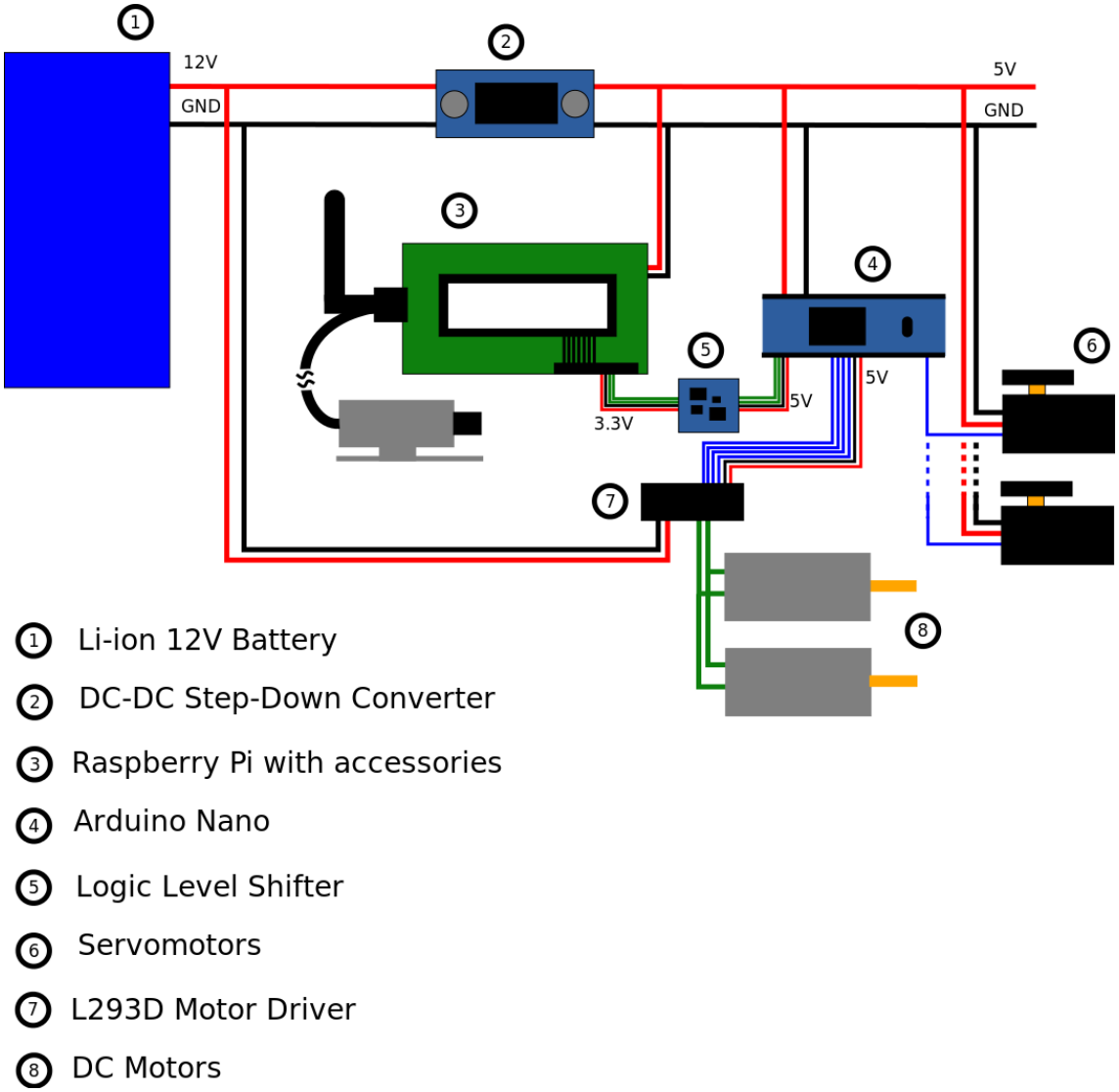

Figure 38: Electrical connections diagram

The previous diagram shows how all components are interconnected, but to increase its clarity some connections have not been shown in detail. The following diagrams show how the remaining elements are wired pin by pin.

- Figure 39 shows the conections between the LCD and Rasbperry Pi and the Raspberry, Arduino and the logic voltage shifter pins.

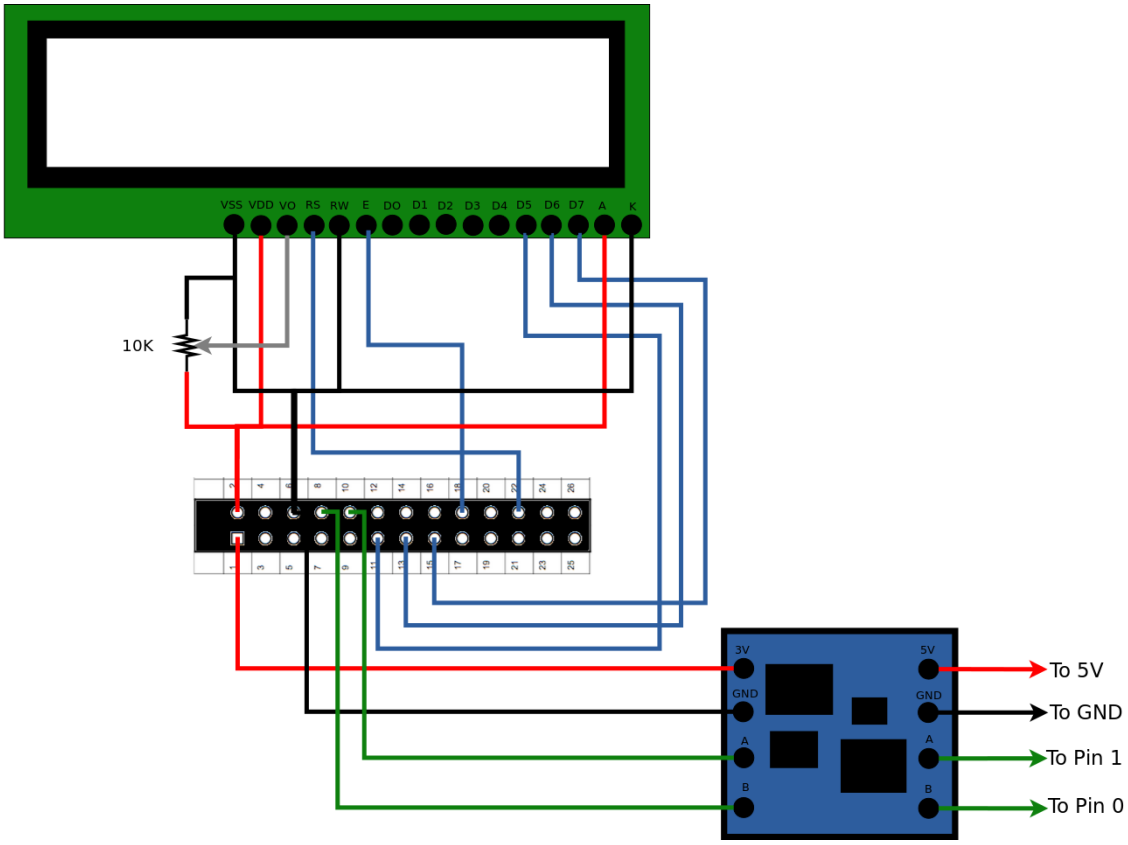

Figure 39: Detail of LCD and Serial connections

- Figure 40 shows the wiring between the motor driver, the motors and the Arduino.

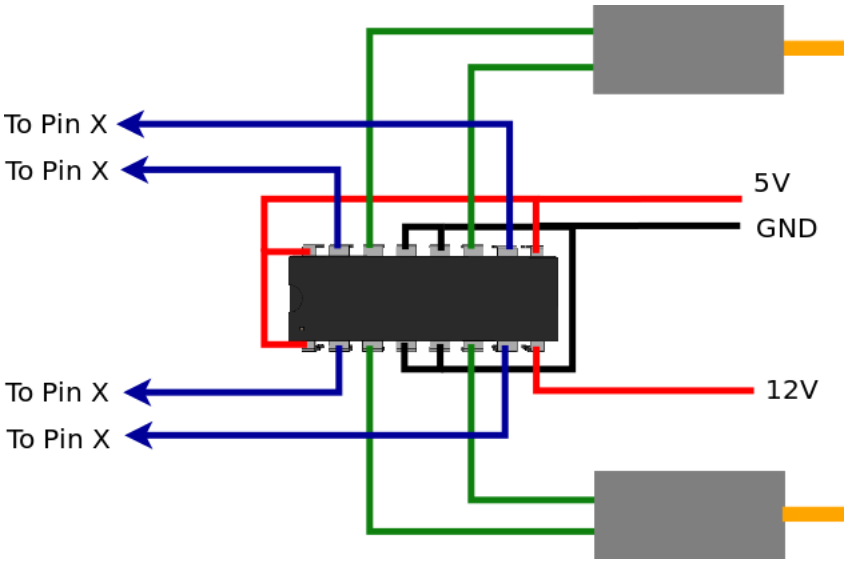

Figure 40: Detail of motor driver wiring

- Figure X shows which element is connected to each of the Arduino pins
- "table with servo +motor letters assigned to arduino pins"

7.2.2 Logic connections

Figure 41 shows how the different components communicate between themselves. As it can be seen, the user controls the robot from the Android application. This implements a bidirectional communication over wifi with the Raspberry Pi, which is used to both send the Raspberry data concerning the movement of the different motors and to receive the video stream from the robot's onboard camera. The Raspberry then communicates over Serial port with the Arduino, which takes care of the data received to obey the user's commands.

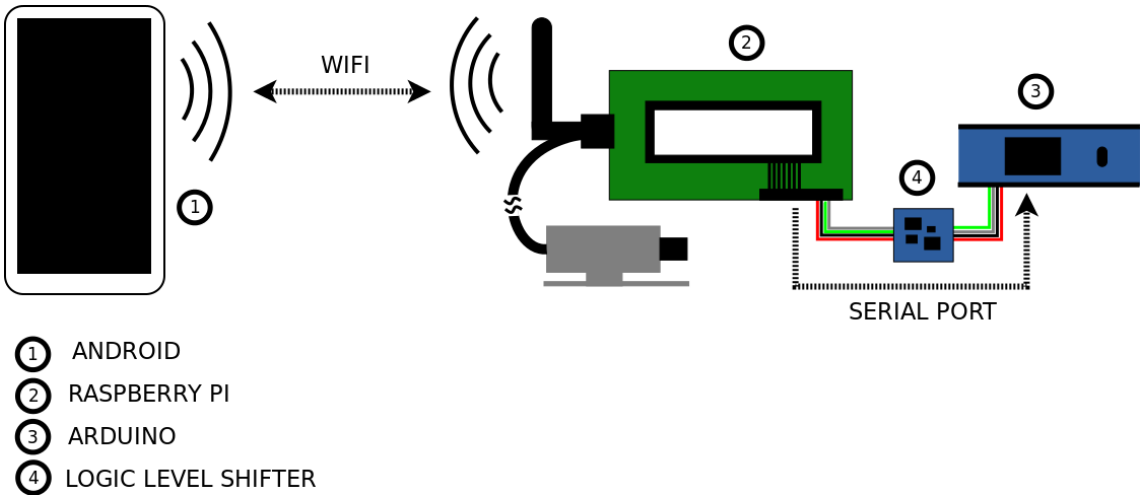

Figure 41: Logic connections diagram

7.2.3 Software connections

Figure 42 shows how the different programs interconnect the various components. It can thus be seen that the Raspberry Pi will first create a wifi network and then start to stream video through it. The android phone on the other hand will connect itself to the recently created network and will use it to emit the commands given by the user. The Raspberry will have already started the IP/UART Bridge, and will send the data received from the phone to the Arduino. Finally, the latter will execute its code to execute the orders received.

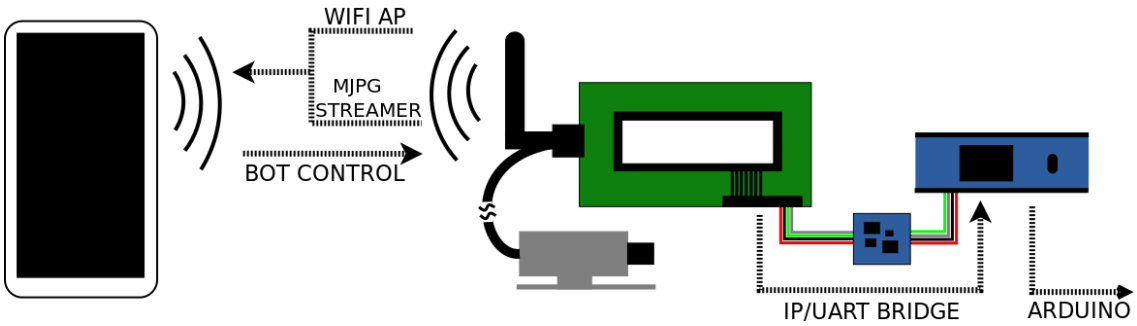

Figure 42: Software connections diagram

8 Arduino

8.1 Overview

Arduino refers both to the microcontroller board used to interface with sensors and actuators and to the software used to program it.

As a microcontroller, an Arduino is a relatively cheap development board useful for controlling many input and output pins, either digital or analog, in a single board solution that plugs directly into the computer over USB.

As a software environment, it provides a simple IDE with many code examples, a bootloader to program microcontroller chips directly with almost no external components and a growing user community that creates libraries for different sensors and communication protocols among others.

All Arduino programs follow the structure presented in Figure 43, namely one Setup function and one Loop function. The first is executed only once at the beginning of the program, while the latter is equivalent to a "while(1)" block, meaning that any code entered into it will be repeated until the microcontroller shuts down or is reset.

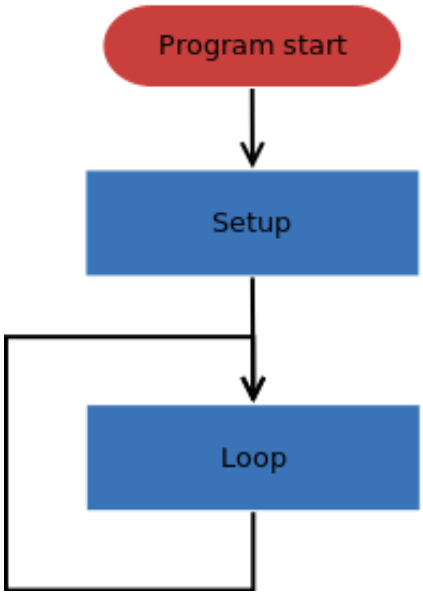

Figure 43: Arduino code skeleton

8.2 Code

In this section the code written for the robot's controller will be explained. A flowchart diagram of the program is illustrated by Figure 44.

- As it can be seen, the Arduino first defines all the robot's data, including the motor, servomotor and communication pins. This ensures the microcontroller knows where to send each datum once the user connects to the robot.

- The program then enters its Loop function. Here it will check if the serial port is available, eg the user has sent a stream of data. Once the port is available, the Read function is called, which stores every byte received into a string to be used later. Once the reading has ended the code checks if it has received a special end-of-line character that signals the end of the data stream. If all the data was retrieved the code moves on to the next function.

- The Parse function is called upon next. This function's purpose is to break and convert the previously stored string into the corresponding variables needed by each element, taking into account their sizes and types. Hence, it transforms one line of numbers into many parameters such as rotation angle, arm selection or movement direction which will be used by the next function.

- With the data correctly formatted, the program executes the Process function which is where the "thinking" is done. Here are defined all the rules the robot must follow, such as knowing which claw to close depending on the side chosen by the user but closing both if the symmetry box was checked. It takes the data provided by the previous function and processes them to end up with a structured list of variables ready to be assigned to each element.

- In the next step the Write function is called. This very simple function goes through the previous list assigning each variable to the corresponding element's assigned pins.

- Finally, the code clears the initial string to make space for new data, resets the flag informing of the correct retrieval from the serial port and proceeds to the next iteration within the Loop function, restarting the process.

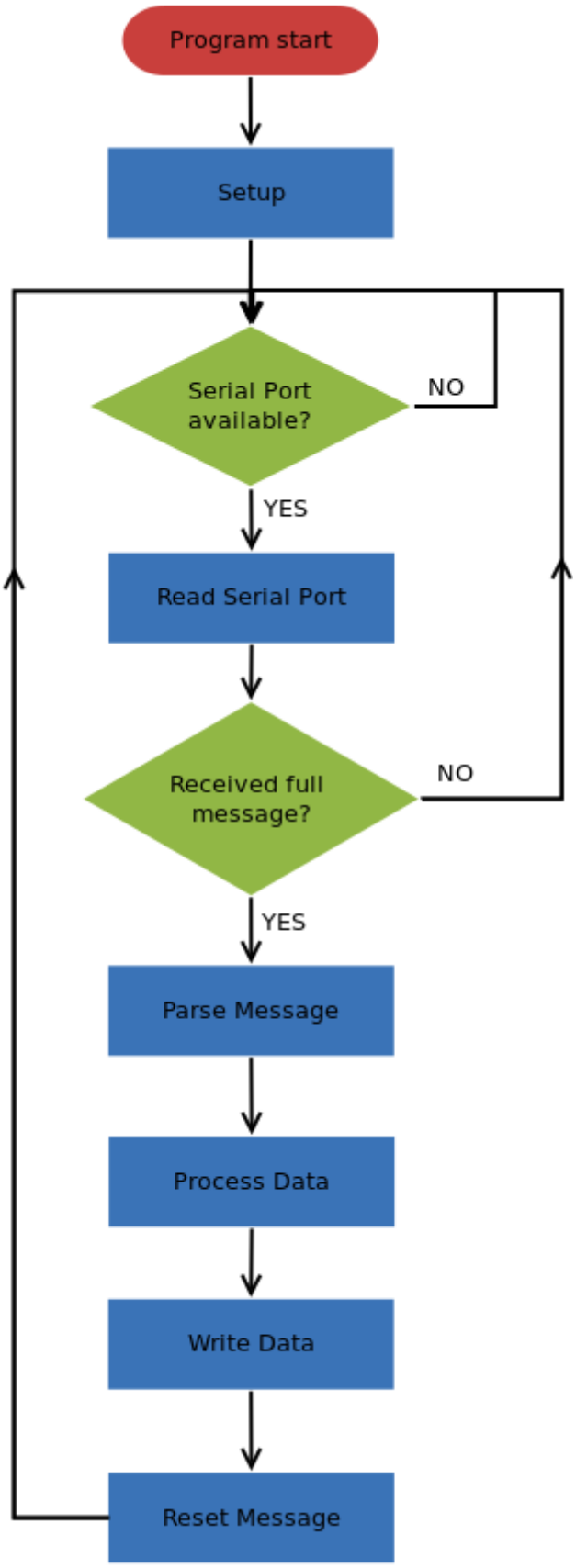

Figure 44: Arduino program flowchart

9 Raspberry Pi

The Raspberry Pi carries out three main duties to ensure everything works correctly. These include creating a wireless connection, streaming images from the camera to the phone and transmitting the data received from the phone to the microcontroller.

These are all placed into the */etc/rc.local* file so the system initializes them automatically each time the robot is turned on, with no need for human interaction.

9.1 Wireless Communications

The chosen means of communication between human and humanoid was wifi. This is so because it is a widely established technology, with great compatibility and in a great number of cases is already installed in the homes of users. It also has a greater speed and range than other technologies, like bluetooth, which are an asset in the case of streaming images.

Three methods were considered: connection to an existing wifi network, creation of an Ad-Hoc connection and establishment of a wifi Access Point.

9.1.1 Existing network

The most straightforward solution is to simply connect the robot to the user's existing wifi network. This enables the user to control it from anywhere in the world, expanding its uses. However, some configuration is required, namely selecting the desired network and introducing the password, which complicates the setup by having to add a keyboard and a display.

This method would thus be suitable for experienced users and developpers, but not necessarily for the average seniors it is intended to help.

9.1.2 Ad-Hoc connection

The next solution implemented was an Ad-Hoc connection between the Raspberry Pi and the Android phone.

This configuration aimed to solve the problem of usability, since the phone would automatically connect to the network, hence eradicating the problem of setting up the communication. This also had the advantage of creating an independent network, and thus being able to operate in remote areas.

In order to create implement this two files need to be set up. Firstly, the computer must be given the specific details of the new network to be created. Here, the contents of Listing 1 must be included into the file */etc/network/interfaces* .

Listing 1: Ad-Hoc Configuration [/etc/network/interfaces]

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
  address 192.168.1.1
  netmask 255.255.255.0
  wireless-channel 1
  wireless-essid RPiAdHocNetwork
  wireless-mode ad-hoc
```

With this configuration the Raspberry will assign itself the IP address 192.168.1.1, but the client computer will be left without an IP assigned and so will be unable to connect to the former.
To provide an IP to the client the package *DHCP Server* must be installed by typing *sudo apt-get install dhcp3-server* into a terminal.

Listing 2 must then be included in file */etc/dhcp/dhcpd.conf*

Listing 2: DHCP Server Configuration [/etc/dhcp/dhcpd.conf]

```
ddns-update-style interim;
default-lease-time 600;
max-lease-time 7200;
authoritative;
log-facility local7;
subnet 192.168.1.0 netmask 255.255.255.0 {
                range 192.168.1.5 192.168.1.150;
}
```

After rebooting the Raspberry Pi, the Ad-Hoc network is created and ready to use. However, while it is compatible with a large range of devices like computers and iOs devices, non-rooted Android devices are not able to connect to the network, which invalidates this procedure as it is not suited for the general public.

9.1.3 Wifi Access Point

The last option for connecting the phone to the robot is to create a wifi Access Point (AP). This method involves a more complex setup than the previous, but while maintaining the same benefits, both allows Android devices to connect and supports speeds of up to 54 Mbps in 802.11g, while the former was limited to 11 Mbps in 802.11b.

The following section will explain how to establish this kind of connection. In order to do so both the Access Point host and the Dynamic Host Configuration Protocol (DHCP) server need to be configured, and will be so following the diagram in Figure 45.

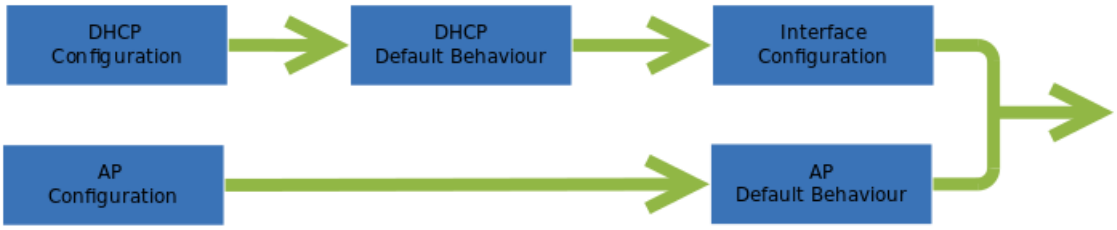

Figure 45: Wifi Access Point connection configuration

Firstly the *Hostapd* and *Isc-dhcp-server* packages have to be installed: *sudo apt-get install hostapd isc-dhcp-server*.

Once the needed packages have been installed the DHCP server must be configured in order to assign IP addresses to clients. The contents of */etc/dhcp/dhcpd.conf* must be replaced with those in Listing 3.

Listing 3: DHCP Server Configuration [/etc/dhcp/dhcpd.conf]

```
# Sample configuration file for ISC dhcpd for Debian
# Attention: If /etc/ltsp/dhcpd.conf exists, that will be used as
# configuration file instead of this file.

# The ddns-updates-style parameter controls whether or not the server
# will attempt to do a DNS update when a lease is confirmed. We default
# to the behavior of the version 2 packages ('none', since DHCP v2
# didn't have support for DDNS.)
ddns-update-style none;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;

# Use this to send dhcp log messages to a different log file
log-facility local7;

subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

The next step is to establish the interface on which DHCP Server should assign IP addresses.

This is done by copying the contents of Listing 4 to the file */etc/default/isc-dhcp-server*.

Listing 4: DHCP Server Defaults [/etc/default/isc-dhcp-server]

```
# Defaults for dhcp initscript
# sourced by /etc/init.d/dhcp
# installed at /etc/default/isc-dhcp-server by the maintainer scripts

#
# This is a POSIX shell fragment
#

# On what interfaces should the DHCP server (dhcpd) serve requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="wlan0"
```

Afterwards, the "wlan0" interface must be set up. In this case any previous configuration will be deleted by replacing the contents of */etc/network/interfaces* with those of Listing 5.

Listing 5: Interface Configuration [/etc/dnetwork/interfaces]

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow hotplug wlan0

iface wlan0 inet static
  address 192.168.42.1
  netmask 255.255.255.0
```

The DHCP configuration is now complete.

The Access Point setup has to be established next. A password-protected network will be created to ensure a secure connection. In this case its name will be "RaspiWifi" and its password "raspberry". Again, the contents of */etc/hostapd/hostapd.conf* should be replaced by those of Listing 6. This file is very sensitive, so no extra spaces are allowed.

Listing 6: AP Configuration [/etc/hostapd/hostapd.conf]

```
interface=wlan0
driver=rtl871xdrv
ssid=RaspiWifi
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Finally, the Raspberry has to be told where to find the configuration file previously created. The file */etc/default/hostapd* must include the contents of Listing 7.

Listing 7: AP Defaults [/etc/default/hostapd]

```
# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about
# alternative methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd
# configuration file and hostapd will be started during system boot.
# An example configurationfile can be found at
#/usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

The only thing remaining is to start the AP service at boot, which is done with the command *sudo update-rc.d hostapd enable*.

This concludes the wireless communications setup, finally using the wifi Access Point method because of the advantages mentioned previously.

9.2 MJPG Streamer

One feature the robot implements is the ability to send a video feed to the user's telephone. This is useful in the case of a patient with limited mobility, as they can navigate it through the rooms of a house without having to follow it.

To achieve this the package *MJPG-streamer* is installed, which captures JPG shots from a camera connected to the computer and streams them as M-JPEG through HTTP to external viewers. It is downloaded from the official repository and built from source using the *GNU Make* utility.

Once built, the package includes three main files: mjpg_streamer, input_uvc.so and output_http.so, with each performing one part of the total procedure. More specifically,

- **mjpg_streamer:** The core of the package, it copies JPG files from an input plugin to one or more output plugins.
- **input_uvc.so:** The input plugin. It is charged of capturing JPG files from a connected webcam.
- **output_http.so:** The output plugin. Its job is to stream the JPG files served by mjpg_streamer according to the M-JPG standard over a HTTP webserver.

The next step is to configure it by giving it the camera's location, the desired resolution and the number of frames per second. All of these parameters should be passed as arguments when calling the program from the command-line, but in order to simplify its initialization, the script in Listing 8 is created.

Listing 8: Streaming Initialization Script [startStreaming.sh]

```
#!/bin/sh
sleep 2
route="/home/pi/mjpg-streamer/mjpg-streamer-code-182/mjpg-streamer"
cd  $route
sudo $route/mjpg_streamer -i "$route/input_uvc.so -d /dev/video0 -n
-f 7 -r QVGA" -o "$route/output_http.so -n -w $route/www"

exit 0
```

The robot now has the ability of streaming images from its webcam, which will be visible on the user's telephone and will give them the ability to control it remotely, even from out of their line of sight .

9.3 IP/UART Bridge

It uses the Adafruit-CharLCD library, which can be downloaded from their repository, to enable writing to the LCD screen.

Figure 46 presents a flowchart of the socket to serial connection software.

The program creates a TCP socket server which continuously searches for clients until one of them connects. Once a connection is secured, the LCD changes from "Awaiting client" to "Client connected" and the program waits instead to receive data from the client. The data received is examined to check if it is a "quit" string, in which case the connection is closed and the program awaits another client. On the other hand, if the data is a valid string from the client, it is passed on through serial communication to the Arduino for it to use.

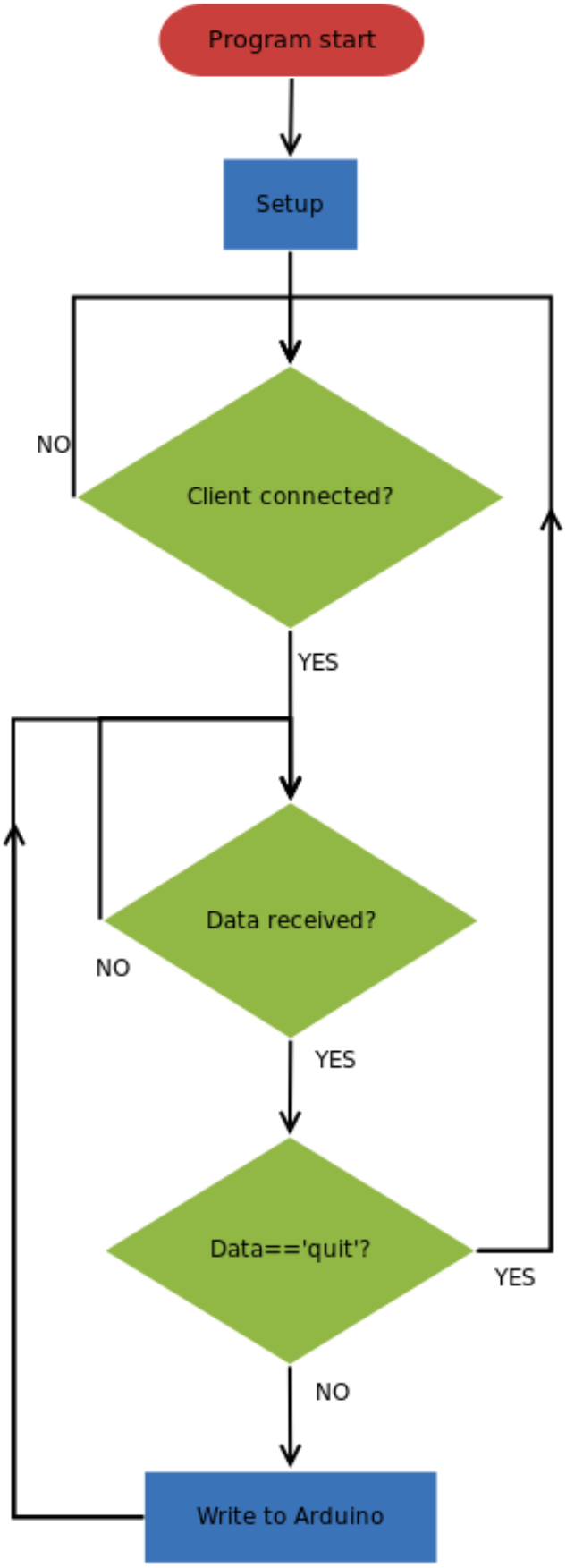

Figure 46: IP/UART program flowchart

9.4 Initializing Script

One of the defining features of this project is that it must be usable by non-technological people, and so it must initialize every service it needs on its own.

To do so, the tasks previously defined are called automatically from a script when the system boots. The contents of */etc/rc.local* are executed right after the computer executes its own routines.

Listing 9: Initialization Script [/etc/rc.local]

```
#!/bin/bash

#Start ip Server
/etc/init.d/isc-dhcp-server start

#Start webcam streaming
#Runs on background so this script is able to launch the next item
#in list
/home/pi/mjpg-streamer/startStreaming.sh &
sleep 0.3

#Start Android to Arduino dumping
/home/pi/AndroidToArduino/startA2A.sh


exit 0
```

The file must be given executable permissions in order to be allowed to implement the commands specified. This is done by typing *sudo chmod +x /etc/rc.local* into a terminal window.

10 Android

10.1 Android Overview

Android applications are typically programmed from Integrated Development Environments (IDE) such as Eclipse or Android Studio. The latter has been used in this because it is the official Android IDE supported by Google.

The typical app skeleton is shown in Figure 47. It consists of the following functions:

- **onCreate()**
  The first function Android calls when the application is launched. Here is where all the static elements are defined, such as setting the views.
- **onStart()**
  This function is called when the app is first visible to the user, after the previous function has ended.
- **onResume()**
  This method is called when the app is ready to interact with the user, ie. it is on top of the application stack and receives the user input.
- **onPause()**
  This is called when a previously started activity is going to be resumed. It is typically used to save data and stop resource consuming parts such as animations.
- **onStop()**
  Called when the activity is no longer visible to the user, either because another activity or the current one is being destroyed.
- **onRestart()**
  It is called when the activity is has been stopped, before it is restarted.
- **onDestroy()**
  If the activity is not restarted it is destroyed. It is the final function used in the activity and is called either explicitly with the *finish()* method or because the system closes it because it is low on resources.

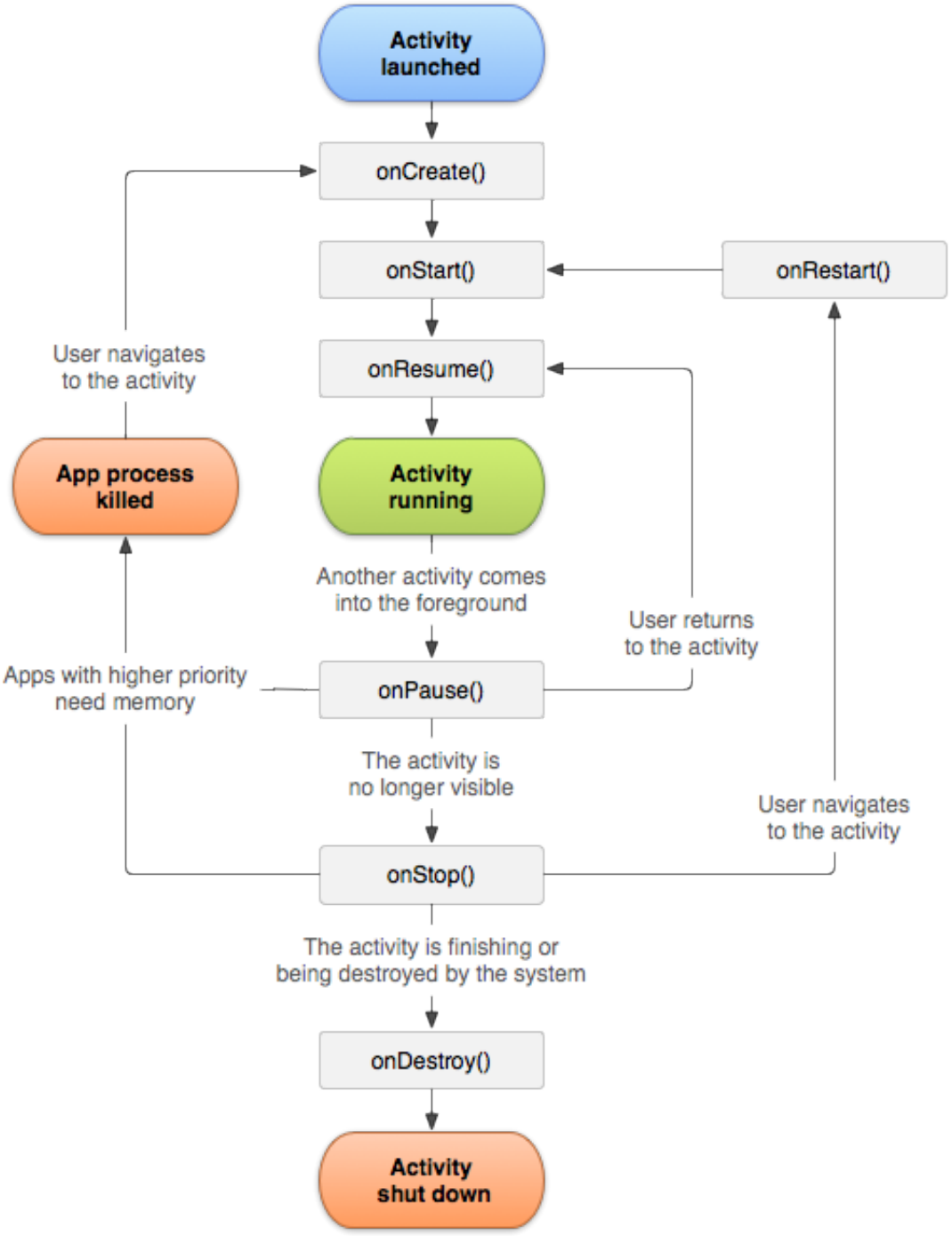

Figure 47: Android app activity lifecycle

10.2 Bot Control

The robot is controlled from the Android application Bot Control, seen in Figure 48. As it can be seen it is divided in two halves, with the upper half displaying the video received from the bot and the lower one encompassing the controls. The complete list of widgets include:

- A WebView connected to the url given by MJPG-Streamer which displays the video feed streamed from the webcam
- Four SeekBars, used to select the desired angle for each of the servomotors that position the arms
- A "Left/Right" Switch for selecting between the left and right arms
- A "Close" Button to close the claw of the current arm
- A "Reset" Button to reset the position of the arms, turning each servo at 90 degrees
- A "Symmetry" CheckBox to activate said option, under which both arms are controlled simultaneously and symetrically
- Five direction Buttons to navigate the robot, which result in it moving forwards or backwards, turning left or right and stopping in place

Figure 48: Bot Control application

The application follows the same general structure presented in the previous section. Here each function used will be analyzed if their default behaviour has been overridden. The following is a list of the modified functions specified in the program's code.

- **onCreate()**
  This is the first function called upon start. The clientThread class is started here and the WebView connects to the robot's IP to reproduce the video stream.

The "Activity running" state is in this case composed by two separate functions, as illustrated by Figure 49.

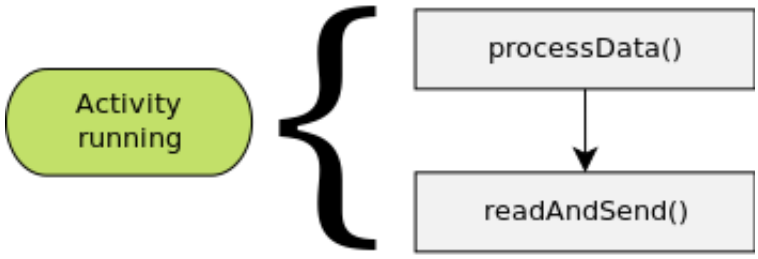

Figure 49: Detail of the "Activity running" state

- **processData()**
  This function identifies and stores changes in movement parameters, such as those resulting in arm or body movement, and calls *readAndSend()* to send them to the robot.

- **readAndSend()**
  This function is in charge of reading the values of non-movement parameters, such as symmetry or side selection, and sends these and the previous to the robot through the socket.

- **onStop()**
  This is the last function to be called before the activity's destruction. It closes the socket after sending the "quit" keyword that tells the Raspberry Pi to start looking for clients again.

- **clientThread**
  The runnable class *clientThread* is in charge of creating a socket client that will be used to send commands over wifi.

In conclusion, Bot Control sends information on the movements the robot needs to deliver while it displays what the latter sees, in order to be able to perform actions like grabbing a bottle from one room and navigating back to the user without having to follow it around the house.

11 Conclusion

12 Future Work

References

[1] EXAMPLE CITATION: Leslie Lamport, *LaTeX: a document preparation system*. Addison Wesley, Massachusetts, 2nd edition, 1994.