BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS
3D PRINTED ROBOT ARM

Supervisor
Lect. PhD Sterca Adrian

Author
Hangan Florin

2019

CONTENT

INTRODUCTION

CHAPTER I

Technologies

1.1 Servomotor

1.1.1 What is a servomotor?

1.1.2 Working principle of Servo Motors

1.1.3 Controlling a servo motor

1.1.4 Arduino

1.2 CAD

1.2.1 Definition

1.2.2 Creating a 3D model in Fusion 360.

1.3 3D Printer

1.3.1 Definition

1.3.2 Construction and parts of a 3D printer

CHAPTER II

Case Study

2.1 Industrial robots

2.1.1 Categories

2.1.2 History

2.1.3 Properties

2.1.4 Interfaces

2.1.5 End effector

2.1.6 Singularities

CHAPTER III

3.1 General Presentation

3.2 Design

3.2.1 Joint

3.2.2 Servos

3.2.3 Control arm

3.2.4 Power management

3.2.5 Control box

3.2.6 Base

3.2.7 Gripper

3.2.8 Circuit board

3.3 Implementation

3.4 User manual

3.5 Improvements

CONCLUSION

BIBLIOGRAPHY

INTRODUCTION

The latest technological improvements from the last decade has brought us to a point where almost anything is possible. The common consumer now has access to a vast majority of information and technologies that were hard to achieve in the past or were pretty expensive.

One of the fields that was hugely improved in the past years is the 3D printing technology. In present days one can afford a 3D printer for a relative low price (compared with the prices over some years ago) and with the help of the internet a huge community was created that helped in reducing the costs and improving this field. The vast number of forums, tutorials and documents helped this field to grow. Now one can easily design, create and print almost everything, from a simple support for a phone to more complex parts like a gear box, or even a jet engine.

CHAPTER I
Technologies

1.1 Servomotor

1.1.1 What is a servomotor?

A Servo Motor is an electrical device that has an output shaft whose angular position can be controlled with great accuracy with the help of an electrical signal. It consists of a motor (usually a DC motor is used), a potentiometer which acts as a sensor to detect the current angular position of the shaft, a gear assembly to reduce the speed of the motor and increase the torque and a control circuit.

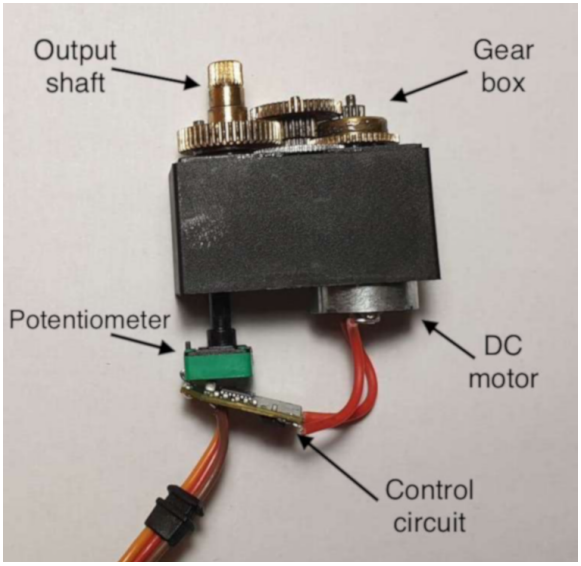

Figure 1.1 Construction of a servomotor

1.1.2 Working principle of Servo Motors

A servo motor is a closed loop system, which means that it uses a feedback system to control the final position of the shaft. Based on the sensor value inside the servo and the input signal an error value is calculated. Based on this error the motor is rotated in such a way that the error is always 0, i.e. the sensor value coincides with the input value. The output shaft is directly connected to the potentiometer, so as the motor is rotated the potentiometer's value changes. After some time, the value of the potentiometer will reach the input signal and in this situation the motor stops rotating.

1.1.3 Controlling a servo motor

All servos have 3 wires, 2 for the power (labeled brown and red) and one for the control signal (labeled yellow). This signal is typically generated by a microcontroller (for example an Arduino). The control signal of a servo motor is a PWM signal (Pulse with Modulation) which means that the angular position is controlled by varying the length of the applied pulse. A typical servo motor can be rotated from 0 to 180 degrees by varying the pulse from 1 millisecond to 2 milliseconds i.e., a pulse of 1 millisecond corresponds to a rotation of 0 degrees, a pulse of 1.5 ms rotates the servo to 90 degrees (neutral position) and a pulse of 2 ms rotates the servo to 180 degrees. The PWM period, i.e. the time between 2 pulses, is 20 ms or 50 Hz. All servos work with a voltage between 4.8 V to 6 V but they can consume a lot of current, up to 2 amperres at intense operations. That is why a separate power supply is mandatory.

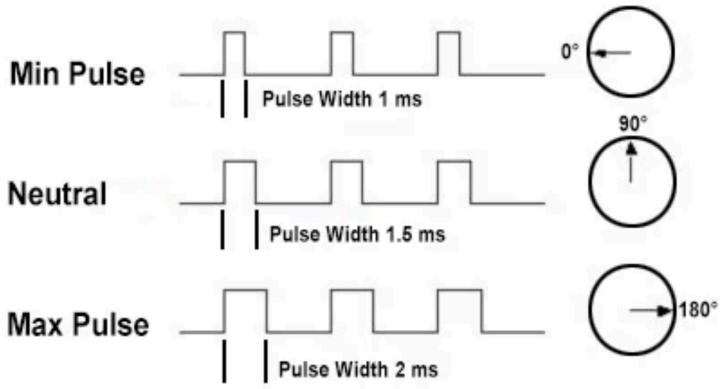

Figure 1.2 Control signal of a servomotor [1]

1.1.4 Arduino

Arduino MEGA 2560 is specially designed for complex projects such as 3D printers and robotics. It has 16 analog inputs and 54 digital I/O pins which give more opportunities for a project.

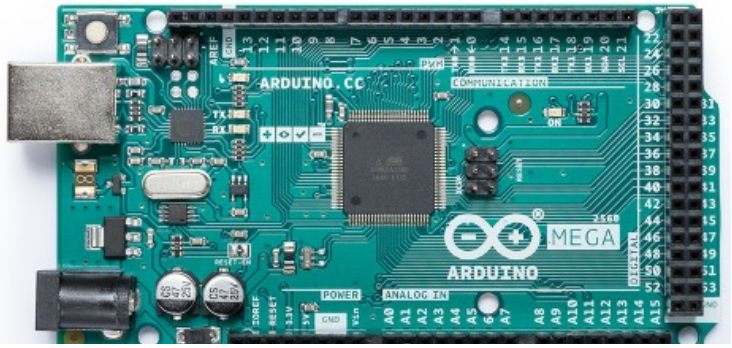

Figure 1.3 Arduino Mega 2056[2]

1.2 CAD

1.2.1 Definition

CAD comes from Computer-aided design and is a computer technology that can help to create 2D precision drawings, 3D models, as well as render, animation or simulations. Some common known CAD programs are AutoCAD, SOLIDWORKS, FreeCAD, Fusion 360, etc. For the arm robot Fusion 360 was used.

1.2.2 Creating a 3D model in Fusion 360.

Fusion 360 is a powerfull tool for creating 3D models. It has lots of features to facilitate the creation of a 3D model in the easiest way possible. Ex: constrains (fixed dimensions, tangent, parallel lines), intersection of objects (one object can be cut using another), fillets (round an edge), etc.

The basic operations for creating a 3D model are creating a sketch and extruding a sketch. A sketch is a 2D drawing of a shape (e.g. rectangles, circles, etc.). A shape can be constrained (have a given dimension). A 3D model can be created by extruding a 2D sketch, i.e. give a thickness to a 2D drawing. Fusion 360 has multiple extrusion types that can add or remove material. By using these two commands, complex 3D objects can be created, for example, this model was created using 6 simple steps:

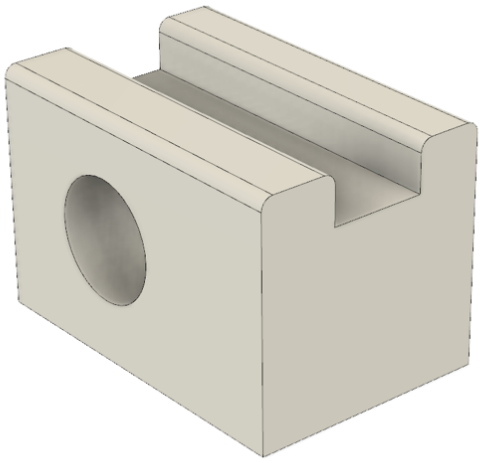

Figure 1.4 Simple model created in Fusion 360

1. A sketch of the base is created with the desired dimensions:

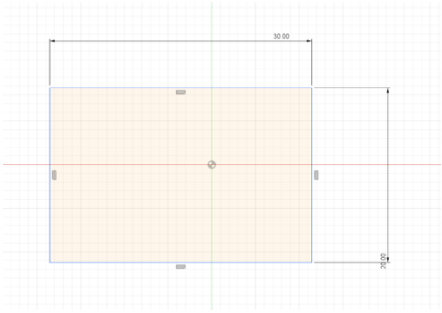

Figure 1.5 Creating a sketch

2. This model is extruded to create the 3D model:

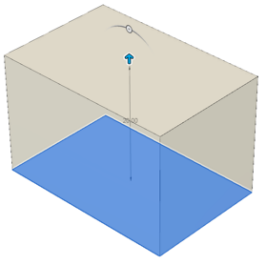

Figure 1.6 Extruding a sketch

3. A new sketch is created on the model with the some dimensions:

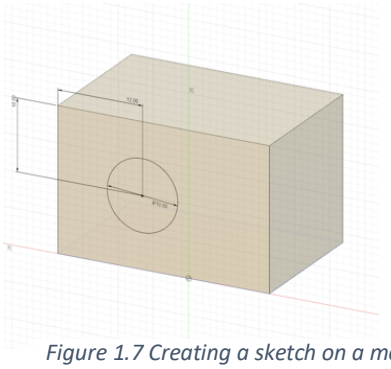

Figure 1.7 Creating a sketch on a model

5. A new sketch is created and extruded using the cut operation:

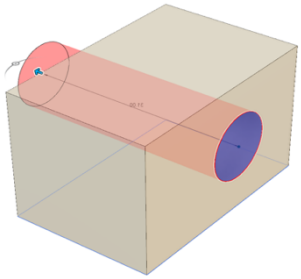

Figure 1.8 Cutting a model

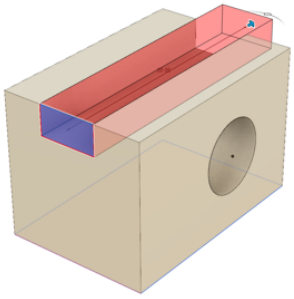

Figure 1.9 Cutting the top of the model

6. At the end a filet operation is used to round the top edges:

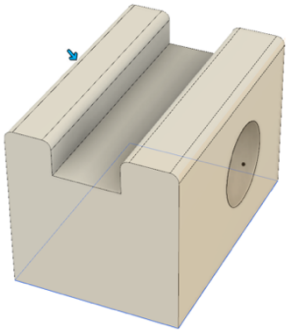

Figure 1.10 Adding filet

1.3 3D Printer

1.3.1 Definition

A 3D printer is a device that can produce 3D solid objects from a digital file. This is achieved using a process called additive manufacturing. An object is created by continuously adding tiny layers of material on top of the object. Each layer is a small horizontal cross-section slice of the object. The most common 3D printers use a cartesian system, that is, each axis corresponds to a dimensional coordinate system: X, Y and Z.

1.3.2 Construction and parts of a 3D printer

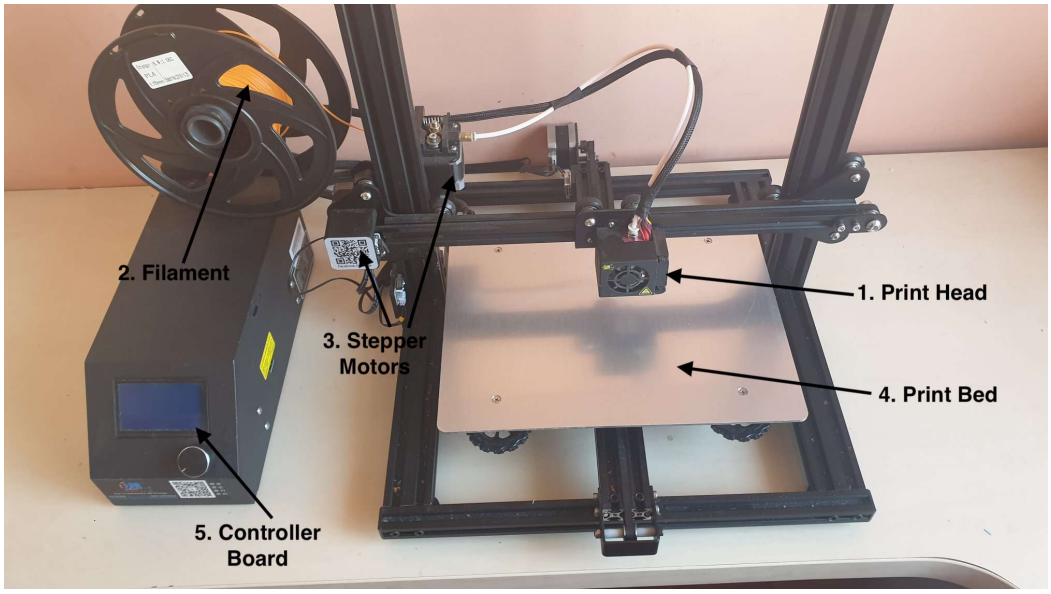

Figure 1.11 Components of a 3D printer

1. Print Head – or the extruder, is the most important part in a 3D printer. Consist of a small nozzle located at the top of the extruder with diameter of only 0.4mm and the heater element. The heater has a typical temperature of 220 degrees Celsius and it melts the filaments and deposits it on the model. In order to quickly solidify the melted filament a fan is used to blow cold air on the top of the object.

2. Filament – The filament is the material used to create objects using a 3D printer. It comes in a long wire of plastic loaded into a spool. A typical spool has a mass of 1 kilogram of filament. There is a vast number of filament types, each with its own advantages and disadvantages. Most common types are PLA and ABS but there are other filaments like Wood, Nylon, PETG, Magnetic, Conductive, even Flexible or filaments that glow in the dark.

3. Stepper Motors – Mechanical movement is achieved using stepper motors. They are all connected to each axis and drive the extruder in X, Y and Z directions. A stepper motor moves in incremental steps and if combined with a stepper driver it can be moved very accurately, this is why they are a great choice for a 3D printer instead of a traditional DC motor.

4. Print Bed - The print bed is the place where the extruder puts the hot filament to form a solid object. A printed bed can be heated or not, while it is not necessary, a heated bed can increase the overall quality of the print and avoid warping issues.

5. Controller Board – Also called the motherboard, is the brain of the 3D printer. It is responsible for reading the files form the SD card and controlling the servos based on these values.

Chapter II
Case Study

2.1 Industrial robots

Industrial robots are automated robot systems which are capable to move in three or more axes. They are used worldwide in a vast number of fields such as painting, welding, product inspection, assembly, packaging and labelling, pick and place, testing and many more.

2.1.1 Categories

Most used configurations for industrial robots are SCARA robots, articulated robots, Cartesian coordinate robots and delta robots. Regarding their autonomy, industrial robots are categorized in 2 subgroups, those which are programmed to repeatedly and accurately carry out the same tasks and those which are flexible as to the tasks they have to perform. Robots from the latter category also use various technologies for visual detections such as visual sensors located at the end effector or even artificial intelligence.

2.1.2 History

In 1937 "Bill" Griffith P. Taylor completed the first known industrial robots, built almost entirely with Meccano parts. The robot was powered by a single electric motor, had five axes of movement, including grab and grab rotation, and it could stack blocks of wood using pre-programmed patters.

George Devol founded the first company that produced robots, Unimation, in 1956. He is also the first person to apply for a robotics patent, in 1954. Robots produced by Unimation had the task of moving different objects from one point to another, reason for which they were also called programmable transfer machines. They were programmed in joint coordinated and used hydraulic actuators. Kawasaki Heavy industries and GKN later purchased the license of the technology at Unimation in order to manufacture robots in Japan and England. Cincinnati Milancron Inc. of Ohio was for a time the only competitor of Unimation, fact which changed when massive Japanese conglomerates began to manufacture mass production industrial robots in the late '70s.

The 'Stanford arm' robot invented by Victor Scheinman in 1969 was a huge step for the robotics industry. It was designed to be fully electric and have 6 axis which gave it the possibility to accurately follow arbitrary paths. Victor Scheinman later collaborated with Unimation, which bought his designs.

Figure 2.1 Stanford Arm [4]

Unimation was bought by Westinghouse Electric Corporation for 107 million dollars in 1984, who later sold it for another company.

Nowadays, this industry consists mainly of Japanese companies which produce robots for industrial purposes in mass quantities.

2.1.3 Properties

An industrial robot is defined by the following parameters:

1. *Number of axis*

Also called the degreed of freedom, they determine the reach of a robot, a plane for two axes, space for three or more. Three more axes are required to control the wrist of a robot, called yaw, pitch and roll.

2. *Working envelope*

Defines the space region which a robot can reach.

3. *Kinematics*

Represent the physical arrangement of a robot's components. This determined the possible motions a robot can make. The kinematics are categorized in multiple class such as cartesian, articulated, SCARA or parallel.

4. *Payload*

Defines the maximum capacity which a robot can lift.

5. *Speed*

Is determined by how fast the end of a robot's arm can be positioned.

6. *Acceleration*

Defines how fast an axis can accelerate. This is also a limiting factor for a robot because may not be able to reach its maximum speed on a short distance or on a long path that requires changes in speed.

7. Accuracy

Measures how accurate is a robot in reaching a commanded position. It can be improved using external sensing such as infra-red sensors. Accuracy can vary for a robot depending on a number of factors such as speed or carrying capacity.

8. Repeatability

Though often confused with accuracy, it defines how well a robot will come back to a programmed position. It is considered to be the most important aspect of a robot.

9. Motion control

It varies depending on the task a robot needs to perform. If a robot has a pick and place continuous task, it only needs to reach the endpoints of the path with accuracy, but if the task resembles that of welding or spray painting, then the movements need to be controlled the whole time.

10. Power source

Electric motors or hydraulic actuators can be used as power sources for a robot, though the former is used more often nowadays due to improvements in protection against possible explosions.

11. Compliance

Measures the distance a robot's axis is moved when a force is applied to it. This happens mostly when the maximum payload is carried.

2.1.4 Interfaces

A robot usually learns the set of movements it needs to perform by a connection to a laptop, a desktop computer or a network. A robot and the collections of machines and peripherals it comes with are controller by a single computer called programmable logic controller. There are two entities that need to be taught: positional data and procedure. The first one refers to the context placement of the objects the robot needs to interact with, and the second refers to how the robot needs to move in order to interact with those objects.

There are techniques used for teaching a robot the positions such as positional commands, teach pendant, lead-by-the-nose, offline programming, robot simulation and others.

2.1.5 End effector

Also called end-of-arm-tooling (EOT), the end effector is the most essential peripheral of a robot. They are often highly complex in order to be able to comply with the handled product. Common examples include gripping devices, belt grinders, spray guns or welding devices.

End effectors may also contain different sensors which help the robot in positioning, handling and locating products.

2.1.6 Singularities

A singularity defines a point in the working envelope of a robot which the end effector is unable to reach no matter how the robot moves its joints. It occurs most often in triple-roll wrist type of a robot's arm.

One such example can occur when the axis 4 and 6 of a robot (first and third axis of the wrist) line up during movement.

CHAPTER III
3D Printed Robot Arm

3.1 General Presentation

This is a 3D printed robot arm that has 4 axis of motion, the movement is achieved using adapted servomotors and is controlled by a smaller arm. The software interprets the angle of every joint from the controlled arm and commands the servomotors from the robot in such a way to reproduce the motion of the small arm. Doing so the control of the robot is made very easy.

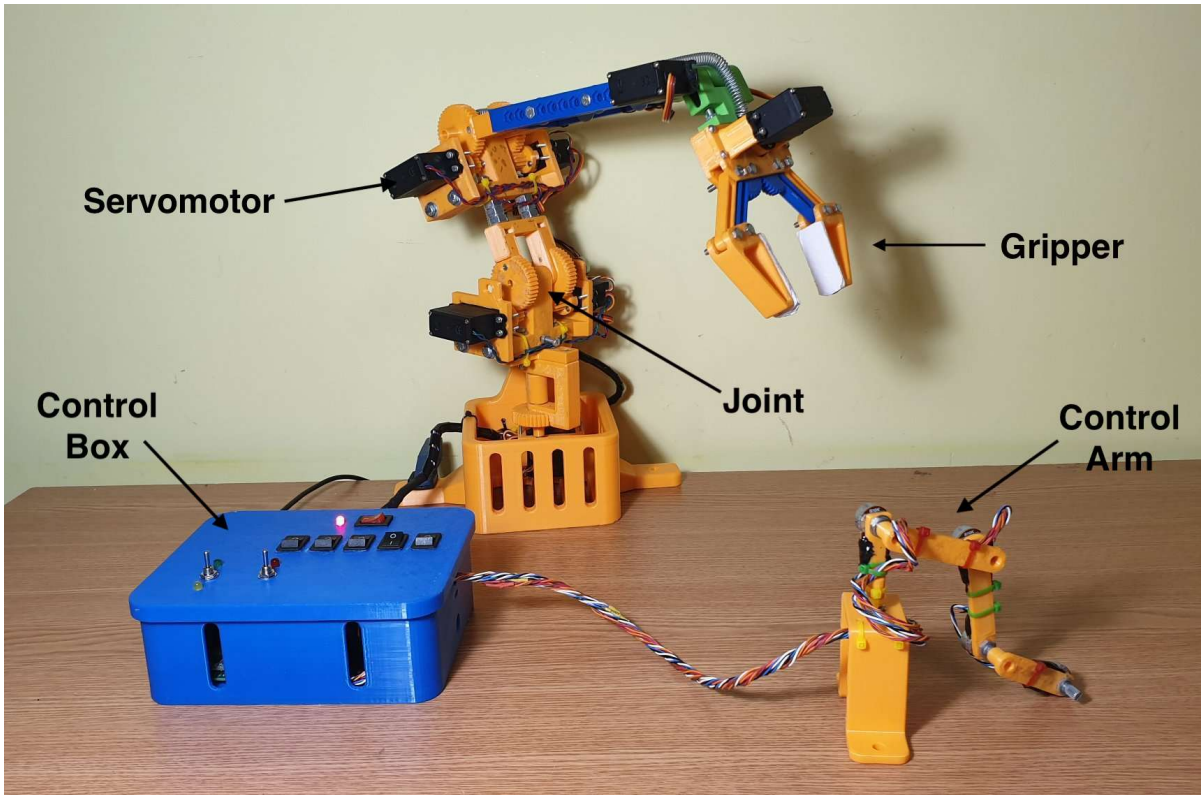

Figure 3.1 General presentation of the robot

Another feature is the ability to record the movements of the robot and to replay them later. The microcontroller used is an Arduino Mega 2056, this was choose for its vast number of GPIO (general purpose input outputs) pins, for its internal memory of 8 kB of RAM and for its clock speed of 16 MHz. By choosing this microcontroller the software can record up to 1000 data points with 50 milliseconds between 2 recorded data which results in a total of 50 seconds of record and play.

The end effector is a gripper which can be used in grabbing different objects. This was designed in such a way that the 2 claws are always parallel to each other, doing so the robot doesn't have to adjust its position when grabbing an object.

3.2 Design

3.2.1 Joints

In essence, a joint consists of 2 3D printed parts that are connected together using a M5 nut and bolt. The nut and bolt were mandatory in order to not put axial pressure on the servomotors, this results in a strong hinge that is easily controlled.

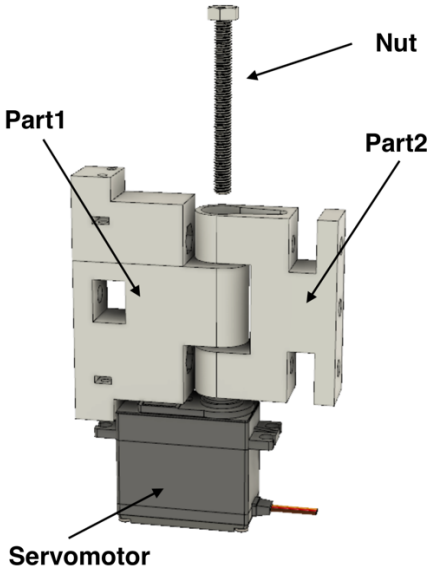

Figure 3.2 Parts of a joint

The design of a joint was made incremental, starting from a simple hinge, a servomotor was added to control the movement, then the design was changed to use 2 servomotors to increase the power and later, because the lack of power was still an issue, the design was radically changed and a gear reduction of 4:1 was added between the servomotor and the joint. This reduced the speed of the movement but had the advantage of increasing the torque. The first 3 joints use the latest design, and the last joint, because it doesn't require a lot of torque, it uses the first design.

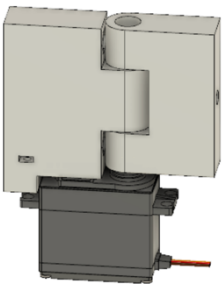

Figure 3.3 First design of the joint

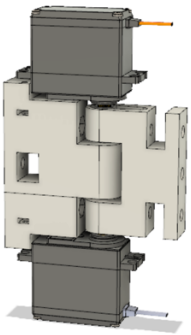

Figure 3.4 Second design of the joint

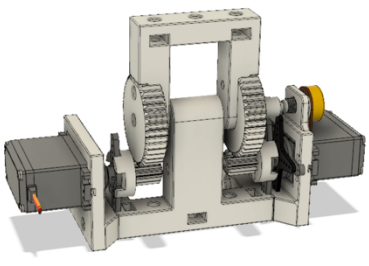

Figure 3.5 Third design of the joint

Because the servos were still pretty weak to lift the whole weight of the arm the solution was to use some springs between joins to hold the arm. There are 2 springs used in this robot, a long one that connects from the base and helps the servos from joints 2 and 3, and a small one connected to the 4th joint. The last spring was helpful because the 4th joint uses only one servomotor.

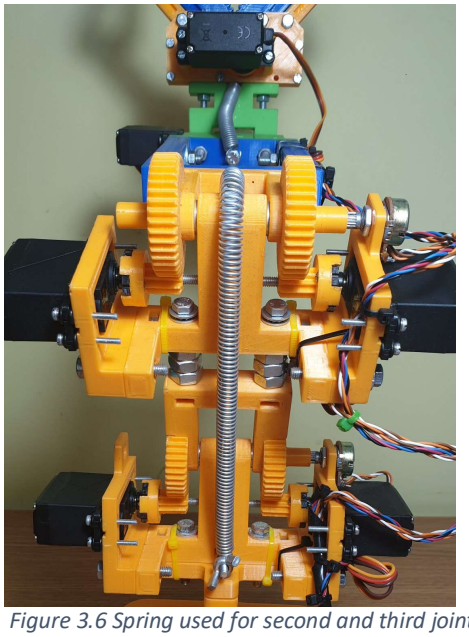

Figure 3.6 Spring used for second and third joint

Figure 3.7 Spring used for the fourth joint

3.2.2 Servos

Servos are the key component for mechanical movement. This robot uses a total of 7 servomotors: 1 for the base, 2 for the second and third joint, only one for the fourth joint and the last one for the gripper.

Because of its internal gear reduction of 294:1 the servo has a great torque but in the context of a robot it has to be able to sustain the whole weight of the arm, thus the torque in this case is not enough, even if two servos were used. The solution was to further increase the torque by changing the gear ratio. This was achieved by designing the joint with extra gears, one for the joint, having 40 teeth and one for the servo having 10 teeth, thus the servo has to make 4 rotations in order for the joint to make one. With the additional gears the overall torque per servo is 1176:1, this means that the small DC motor inside the servo has to make 1176 complete rotations in order for the joint to make only one. Furthermore, to even increase the power, 2 servos were used per joint. One disadvantage is that by increasing the torque the speed is decreased, now it is 4 times slower compared with a joint that is directly connected with a servomotor. Despite this decrease in movement, the tests showed that the arm is still fast enough for normal operations.

In order to be able to add extra reduction, the servos had to be modified. One disadvantage is that a servo is limited to a complete rotation of 180 degrees. The problem is with the internal potentiometer, a rotation greater that 180 degrees results in the destruction of the potentiometer. Furthermore, the joint now has to have some kind of feedback to know its position. The idea was to remove the internal potentiometer from the servo, adjust the servo to allow 360 degrees of freedom (this was done by removing the pin from the output shaft that limits the degree of rotation), add an external potentiometer directly to the joint and connect this external potentiometer to the input feedback inside the servo. Now the servo uses this potentiometer to adjust its position.

Because of the torque issues, 2 servos had to be used. Having an additional servomotor to a joint added extra complexity. Because they are oriented in a symmetrical manner, the signal had to be mirrored for one of the servos, this added extra complexity in the code, wire management and circuit diagram. A solution to this problem was to modify one of the servos to work in reverse. This was done by reversing the leads from the internal potentiometer as well as reversing the DC motor poles. This approach reduced the overall complexity but, because of the inaccuracy of these types of servos, using the same signal for both servos is not a good idea because it is difficult to align them together. Even a slightly offset ended up in jamming them (one servo fought the other one). The final solution was to use only one servo as the control and use the signal from its internal motor to drive the motor from the second one. With this approach both servos work in an interdependency fashion, no jamming is present and the overall complexity was reduced significantly.

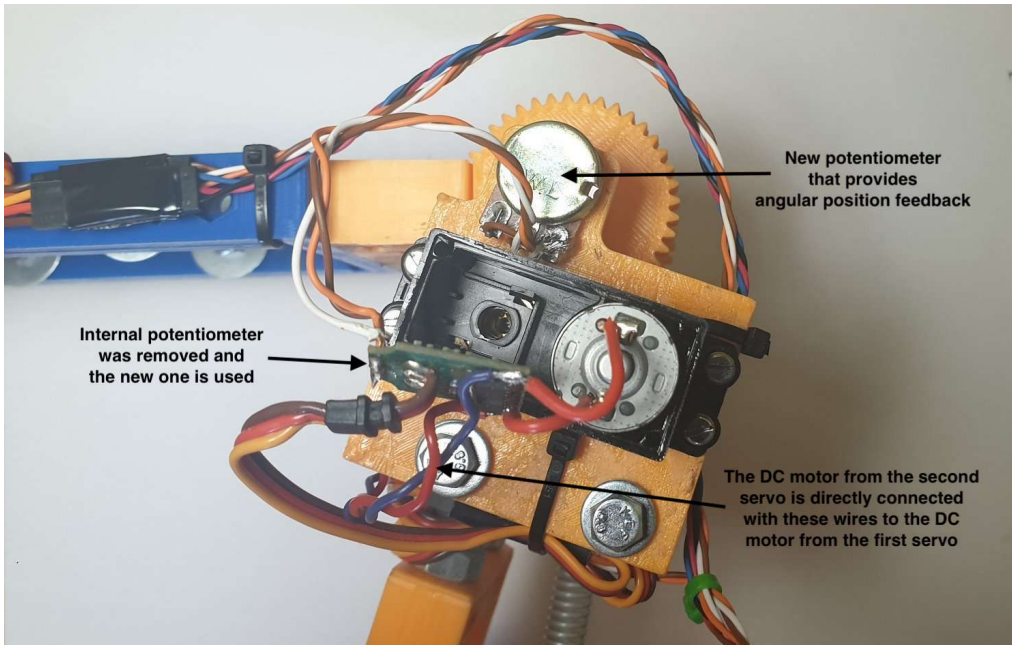

Figure 3.8 Adapted servomotors for the joint

3.2.3 Control arm

The control arm, also called the pot arm, is an exact replica of the robot and it is used to control it. It has 4 joints that are constructed similarly to the robot so the movement of the robot corresponds to the movement of the pot arm. Also, it has another input at the end to control the gripper.

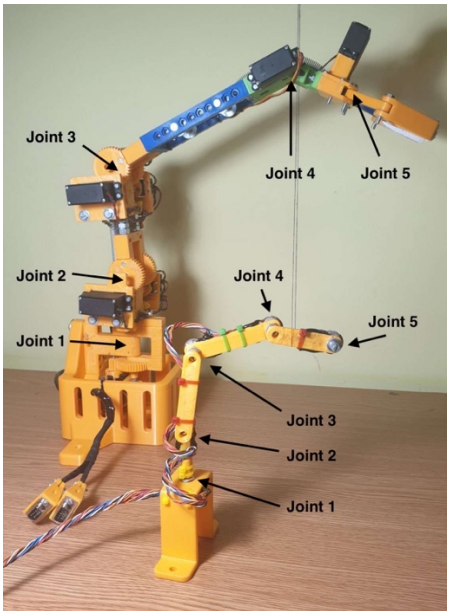

Figure 3.9 Pot arm

The construction of the pot arm is fairly simple, it consists in 5 potentiometers that are mounted with some 3D printed rods. Every hinge is a potentiometer that records the angle of the joint and sends this data to the controller where this information is processed and the right servomotor from the arm is moved in such a way to imitate the angle of the pot arm. For the potentiometers 3 wires are necessary: ground, 5 volts and a signal wire so in total 15 wires are necessary. The wires have a proper connector at the end to easily insert it into the main board. In order to firmly fix the potentiometers and to avoid any signal errors, due to the movement of the pot arm, some hot glue was used to firmly fix the leads. The cables were routed in such a way not to block the movement of the arm and they were fixed with some zip ties.

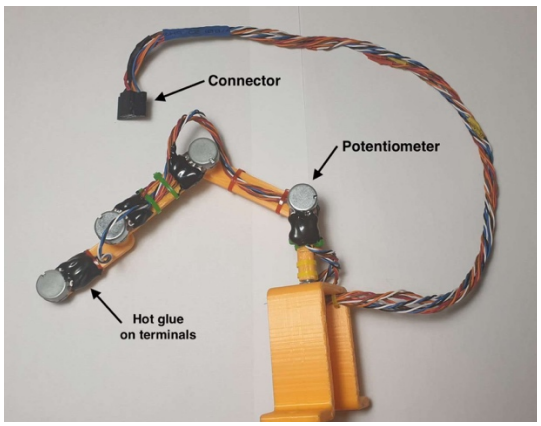

Figure 3.10 Pot arm – back side

3.2.4 Power management

A typical servomotor requires around 4.8 to 6 volts and it draws around 0.5 to 1 amp of current but the peak current can go as high as 2 amperes. When using multiple servos in an application a good power source is mandatory.

Even with a good power source, having 7 servos connected together can cause problems. One common problem is that if one servo draws a lot of current, even for a short amount of time, it will cause a drop in the main voltage and all the servos will become irresponsive and uncontrollable, the only way to reset the servos in this case is to cut all the power and move them a bit by hand. This sort of behaviour is unacceptable and this can lead to the possibility of damaging the servos or even the whole arm. To solve this problem each servo has to be equipped with its own power supply and in case of a failure only that servo should be affected.

To manage the power across all servos, the robot uses 5 buck converters. A buck converter (or a step-down converter) is a circuit that allows stepping down the voltage from its input supply to its output supply. The circuit is a classic switch mode power supply (SMPS) that tries to maintain a fix voltage supply to the output. The model used in the robot is a MP1584EN which is capable of providing up to 3 amperes of current continuously, has an input voltage between 4.5 V to 28 V, an output voltage between 0.8 V to 20 V and an efficiency of 96%, thus no heat sync or cooling method is needed. The output voltage can be set with a small potentiometer located on the board of the converter. Inside the robot, one buck converter is used to power the Arduino and the rest of 4 are used to power the servos. The 4th one powers both the 4th servo and the 5th servo. This is acceptable because the last 2 servomotors do not require a lot of power. Each servo requires a voltage up to 6 V but in this application every servo is overdrive, that is, each buck converter in this case is set to 6.5V. This ensures that the servos have enough power to move the whole arm of the robot. Furthermore, by using a separate power supply for each servo, different voltages can be set for different parts of the robot, for example the servos from the base can be overpowered even more if the arm is not strong enough.

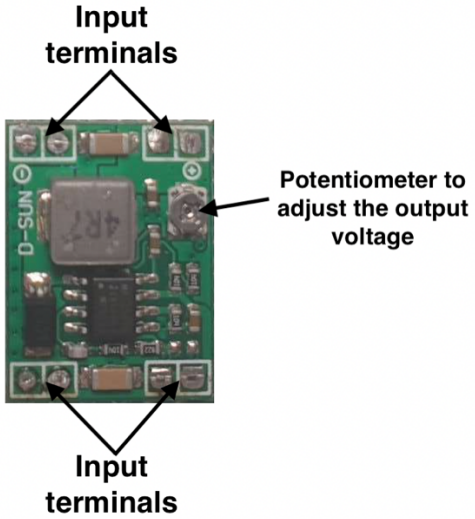

Figure 3.11 MP1584EN buck converter

Because everything is powered through a buck converter the input voltage doesn't have to be fixed, the only requirement is to not exceed the maximum rating of the voltage regulators (28 V) and to have enough power to ensure a good working of the servos and electronics. In this case a laptop power supply is used, this delivers 19 voltages and up to 6 amps, which is enough for this application. The recommended minimum power supply is 12 V at 3 ampere or 9 volts at 5 amperes.

For testing purposes, every buck converter has a switch located on the control box that can interrupt the power to its servo. This feature comes in handy if there is a power problem to one of the servos and it needs to be disconnected from the power supply.

3.2.5 Control box

The control box holds all the electronics, power distribution, the microcontroller and the control switches, all in one 3D printed box. There are 4 connections on the side of the box: the power in, two D-sub connectors that connect to the robot and one programming port for the microcontroller, the connection for the pot arm is made directly on the circuit board. It has a lid that was designed to be press fit into the box and on the lid there is the power switch with the red LED, the 5 switches for every servo and 2 more single pole double throw switches with their LED indicators. Except the 2 SPDT switches which are connected with a bushing and a nut, the switches and LEDs are fit to the lid with super glue.

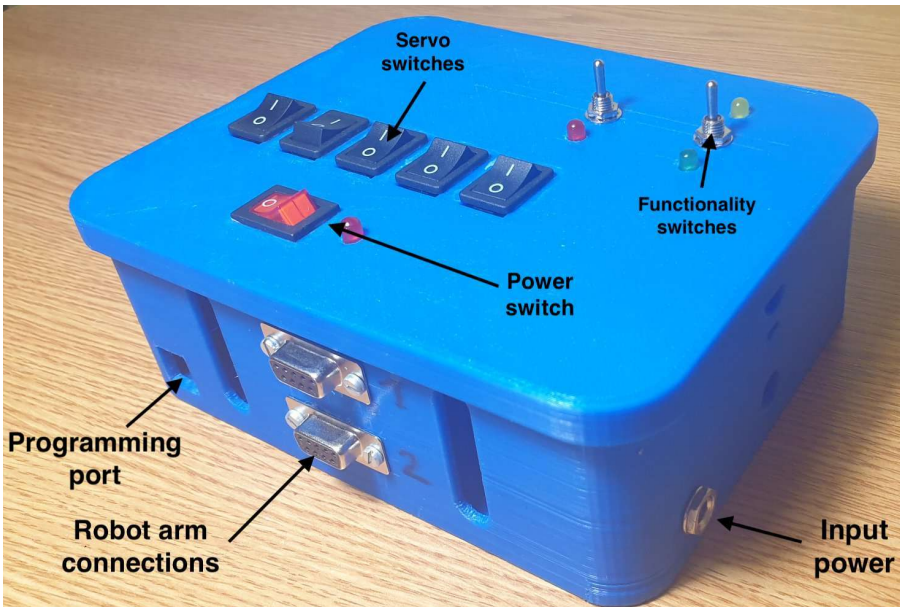

Figure 3.12 Control box

Inside the control box there is the circuit board, the microcontroller and all the necessary connections. All the wires inside the case are long enough so that the lid can be opened to inspect the circuit while still operating the arm and they are designed so that they can be easily disconnected from the circuit board and the microcontroller in case that debugging is necessary.

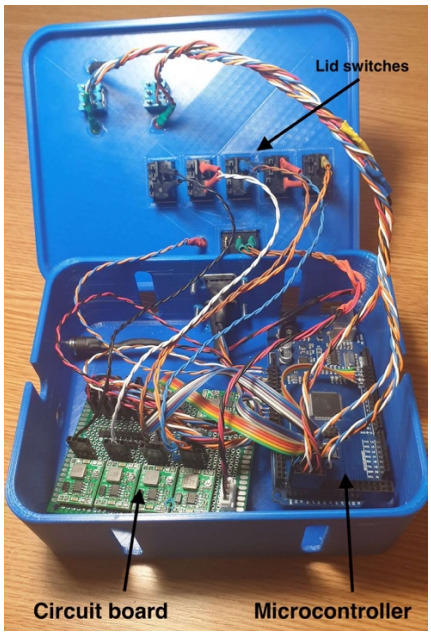

Figure 3.13 Control box – inside

This piece was designed with holes for good ventilation and to allow the cables from the pot arm to easily be inserted and fixed to the box using some zip ties. The lid was designed with the holes for the power switches, the rest of the wholes (for LEDs, SPDT switches and programming port) were added later using a drill. The total time for the box and the lid to print was **17 hours**.

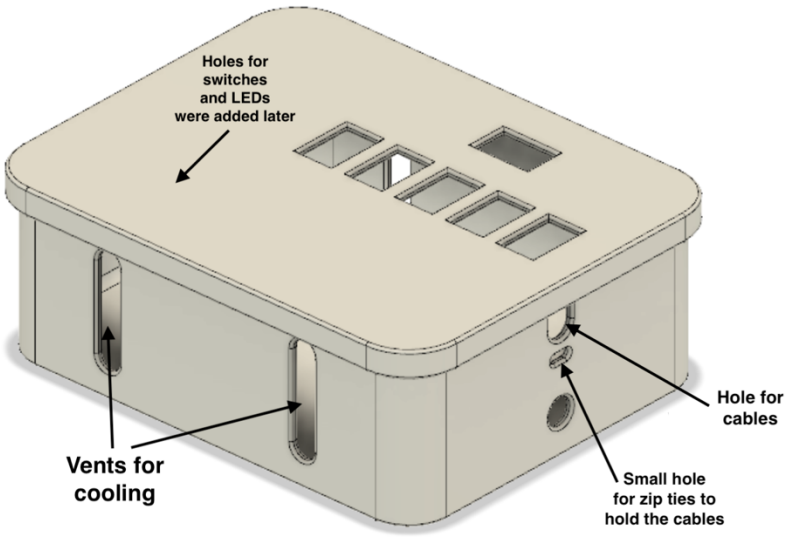

Figure 3.14 Design of the control box

3.2.6 Base

The base of the robot is a 3D printed part that holds the robot. This piece has to be strong enough in order to sustain the whole weight of the arm. It has 2 legs and with their help the robot can be securely fixed to a table or a working bench. The legs have a 7 mm hole which allows for the robot to be fixed with 2 nuts and bolts also. Because it is fairly big, the total printing time for this piece was **23 hours**.

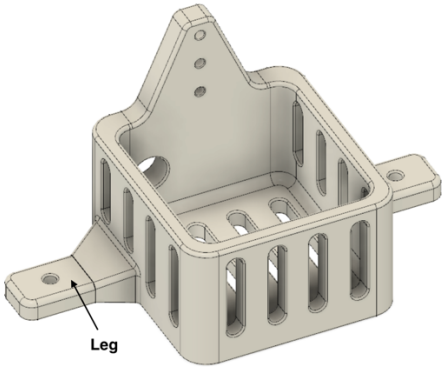

Figure 3.15 Design of the base

3.2.7 Gripper

The end effector of this robot is a gripper. Its purpose is to easily pick and place objects. It was designed so that the 2 claws are always parallel to each other and are always symmetric to the gripper. This was done by using some cogwheels and some parallel rods. It uses a servomotor with a small cogwheel and it was design in such a way that a full rotation of the servomotor corresponds to a full open and close of the gripper. By using a smaller gear there is an additional improvement into the overall torque of the gripper, thus it can easily pick various sizes of objects and hold them tight. Also, the claws have a special material on them to improve grip and to reduce stress on the objects that are being picked.

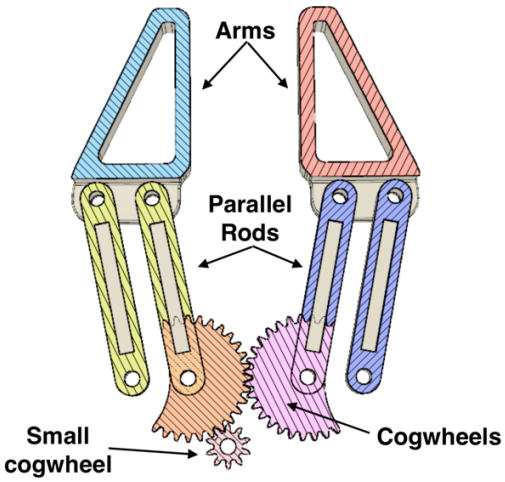

Figure 3.16 Cross section of the gripper

3.2.8 Circuit board

The circuit board bonds all the connections from the pot arm, the robot, the microcontroller, the power distribution and the switches together. It was designed on a double sided protoboard and all the connections were made using soldering. Soldering is a process where 2 metal conductors are joined permanently by melting a filler metal into the joint. The tool used to do this is called a soldering iron or station. This has a metal end that heats and melts the filler metal so that it can flow into the joint between two workpieces.

The board is a single unit, this means that all the connections are inside the board and all wires can be easily disconnected from the board. This design allows for the board to be easily taken out of the control box. All the wires are connected using male and female connectors. The top of the board contains the buck converters, and all the necessary connectors. The bottom of the board contains permanently soldered wires that join together all the connections.

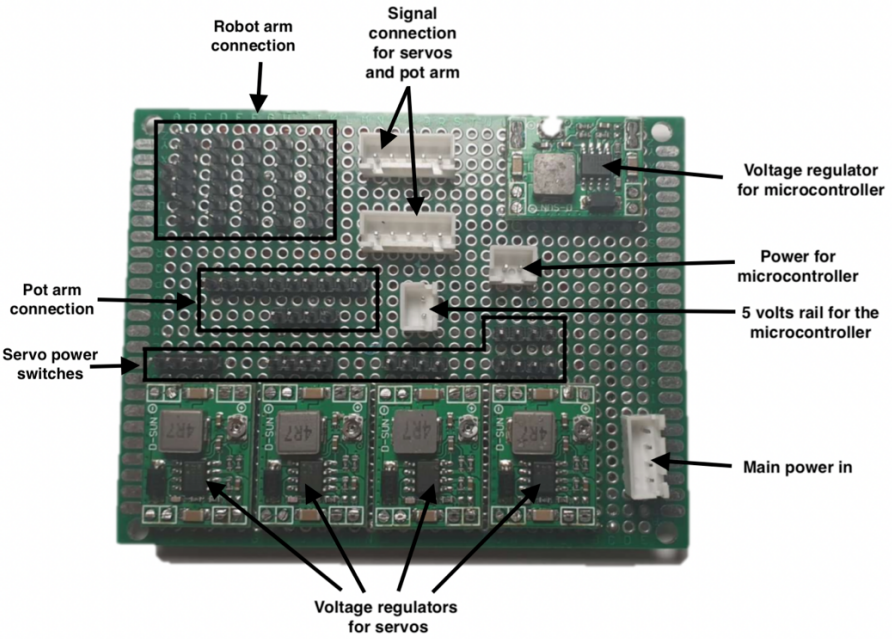

Figure 3.17 Circuit board – top view

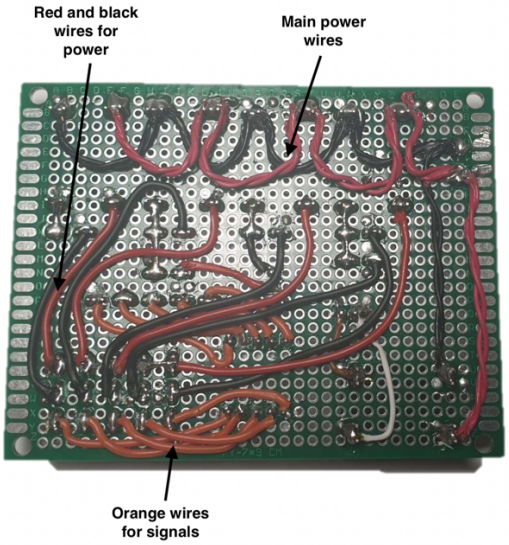

Figure 3.18 Circuit board – bottom view

3.3 Implementation

The code for the robot is written in Arduino, which is similar with the C programming language. The code itself is fairly easy, it contains a single .ino file and has approximately 200 lines of code.

The robot has 3 modes of operation, notated with mode 0, mode 1 and mode 2:

Mode 0 – This is the default mode of operation. In this mode the microcontroller reads the analogue voltages from the pot arm and sends these values to the servos. Because the ADC (analogue to digital converter) of the Arduino is on 10 bits, the values read from the analogue pins goes from 0 to 1023. Because a servo accepts only values between 0 and 180 the data read from the pot arm has to be mapped to the proper values. Arduino has an internal function called map that accepts 5 parameters, the input value, the bounds of the input value and the bound of the desired output value. Furthermore, in order to achieve perfect synchronization of the pot arm with the robotic arm, these values are slightly modified. For example, the values from the first joint were mapped from -10 to 210 degrees. In order to maintain the maximum values allowed for the servos a constraint function was used to limit the final output between 0 and 180 degrees.

Mode 1 – This is the record mode. Every 50 milliseconds the program stores the current position of the arm into a vector called recorded_data. The vector is currently set to a maximum of 1000 data points, this results in a total of 50 seconds of recording.

Mode 2 – This is the play mode. Every 50 milliseconds the program commands the arm to move to the current position saved in the recorded_data. After the position reaches the end of the vector it starts to decrease to the first position. By doing this, sharp movements from the last position to the first position are avoided.

In order to set the corresponding mode, a SPDT switch is used, the middle position is for mode 0, the bottom position is for mode 1 and the top position is for mode 2. For mode 1 and 2, two LEDs are used, located at the top and the bottom of the switch. For mode 1 a yellow LED is used and a green one for mode 2. There is an additional switch that enables or disables the robot, that is, if disabled, data is no longer sent to the servos and they stop. This feature is necessary if the pot arm is not hold in hand and the robot needs to be stopped. Also, this feature is used in mode 2 when the arm has to be stopped and resumed later. The arm is enabled when the switch is in the up position. A red LED, located at the top of the switch, is turned on when the arm is enabled.

To avoid using an external pullup resistor for the switches, the pins are configured to INPUT_PULLUP, this means that the internal resistor from the board is used to pullup the input. By using this mode, the default value read from the pin (i.e. when the pin is left unconnected), is HIGH. To set the pin to LOW the other wire from the switch has to be connected to ground. One small disadvantage is that the values read in the code are 1 when the switch is off and 0 when it is switched on. Although this is not a problem, it needs to be taken care of it in the code.

In the first part of the code all pins that are used are defined, as well as all the necessarily variables and functions.

The setup() method is a function that it's executed only once, when the Arduino starts. In this function the servos are initialized with the correct pins for each servo and the pin mode of the pins used are set (OUTPUT for LEDs and INPUT_PULLUP for the switches). Because of the lack of ground pins on the board, regular GPIO pins were used for the negative terminals of the LEDs. These pins are set to LOW.

```
void setup() {
      //Set the serial. Used for debugging
       Serial.begin(9600);

      //Initialize all servos with the correct pins
      servo1.attach(12);
      servo2.attach(11);
      servo3.attach(10);
      servo4.attach(9);
      servo5.attach(8);

      //Sets all pin modes for switches and leds
      pinMode(LED_PIN_GREEN, OUTPUT); //green led +
      pinMode(44, OUTPUT); //green led -
      pinMode(LED_PIN_YELLOW, OUTPUT); //yellow led +
      pinMode(48, OUTPUT); //yellow led -
      pinMode(LED_PIN_RED, OUTPUT); //red led +
      pinMode(49, OUTPUT); //red led -
      pinMode(SW_PIN_MODE1, INPUT_PULLUP); //sw mode 1
      pinMode(SW_PIN_MODE2, INPUT_PULLUP); //sw mode 2
      pinMode(SW_PIN_ENABLE, INPUT_PULLUP); //sw en

      //Set the negative pins of the leds to LOW
      digitalWrite(44, LOW);
      digitalWrite(48, LOW);
      digitalWrite(49, LOW);
}
```

“After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond.”[9] The loop functions in this case reads the data from the pot arm, reads the switches to determine the mode and the enable functionality then it uses a switch statement for each mode of operation. In mode 0 only the SetServos() function is called which sets the servos with the values read from the pot arm. For mode 1 the SetServo() function is also called in order to still be able to move the robot and then every PERIOD of time the SaveData() is called. To execute a block of code after a period of time the millis() function is used, this returns the time since the board was powered, in milliseconds. For mode 2 the SetServosFromData() is called every *PERIOD* of time, in this case every 50 milliseconds.

```
void loop() {
	ReadPotArm();
	ReadMode();
	ReadEnable();

	switch (mode) {
		case 0: {
			// Only sets the servos with the
			// values from the pot arm
			SetServos();
			break;
		}
		case 1: {
			SetServos(); // Sets the servos
			//Every PERIOD of time save a new pint in
			// recoreded_data
			if ( millis() - prev_time >= PERIOD ) {
				prev_time = millis();
				SaveData();
			}
			break;
		}
		case 2: {
		//Every PERIOD of time set the servos with the
		//current data from recorded_data
			if ( millis() - prev_time >= PERIOD ) {
				prev_time = millis();
				SetServosFromData();
			}
			break;
		}
	}
}
```

ReadPotArm() function reads the data for analog pins using the analogRead() function, maps the values to the correct limits, constrain these values between 0 and 180 degrees and saves this values in the servo_value variable for each servo. The output values form the map are adjusted so that the robot imitates the pot arm precisely. For servo_value4 the output values from the map functions were reversed (i.e. the low value is 180 and the high value is 0), this was done because, during the construction of the pot arm, the potentiometer leads were swapped.

```
void ReadPotArm() {
	//Read every value from the analogue pins and map it to the
	//correct values. Also apply a constrain between 0 and 180 to make
	//sure that the data passed to the servos is a valid angle
	servo_value1 =
constrain(map(analogRead(A1), 0, 1023, -10, 210), 0, 180);
	servo_value2 =
constrain(map(analogRead(A2), 0, 1023, -10, 210), 0, 180);
	servo_value3 =
constrain(map(analogRead(A3), 0, 1023, 210, -10), 0, 180);
	servo_value4 =
constrain(map(analogRead(A4), 0, 1023, 180, 0), 0, 180);
	servo_value5 =
constrain(map(analogRead(A5), 0, 1023, 0, 180), 0, 180);
}
```

SetServos() function updates the servos. To do this the write function is used which expects a value between 0 and 180. Also the enable variable is checked and if it is 0 the servos will not be updated.

```
void SetServos() {
	//Set all servos with the values from the pot arm
	if ( enable ) {
		servo1.write(servo_value1);
		servo2.write(servo_value2);
		servo3.write(servo_value3);
		servo4.write(servo_value4);
		servo5.write(servo_value5);
	}
}
```

ReadMode() function reads the values from the switch and based on these values the mode is calculated. Because the pin mode for these pins were set to INPUT_PULLUP the values are in reversed (i.e. 0 means that the switch is opened and 1 means that it is closed). Depending on the mode, the correct LEDs are turned on: for mode 1 the yellow LED is turned on, for mode 2 the green one is turned on and for the mode 0 both are turned off. Because the values for the current recorded data dimension and position.

```
void ReadMode() {
	//Read the 2 inputs and based on them sets the mode
	//Also turn on the corresponding LEDs
	int v1 = digitalRead(52);
	int v2 = digitalRead(50);
	if ( v1 == 1 && v2 == 1 ) {
		last_mode = mode;
		mode = 0;
		digitalWrite(LED_PIN_GREEN, LOW); //turn off both leds
		digitalWrite(LED_PIN_YELLOW, LOW);
	}
	else if ( v1 == 0 && v2 == 1 ) {
		if ( last_mode != 1 ) { //Reset position and dimension
			position = 0;
			dimension = 0;
		}
		last_mode = mode;
		mode = 1;
		digitalWrite(LED_PIN_GREEN, LOW); //turn on yellow led
		digitalWrite(LED_PIN_YELLOW, HIGH);
	}
	else if ( v1 == 1 && v2 == 0 ) {
		last_mode = mode;
		mode = 2;
		digitalWrite(LED_PIN_GREEN, HIGH); //turn on green led
		digitalWrite(LED_PIN_YELLOW, LOW);
	}
}
```

ReadEnable() function reads the value of the second switch and sets the enable variable. Also turns on or off the red LED indicator.

```
void ReadEnable() {
	//Read the enable value and set the red LED
	enable = digitalRead(SW_PIN_ENABLE) == 0 ? 1 : 0;
	digitalWrite(LED_PIN_RED, enable);
}
```

SaveData() function saves a new point into the *record_data* vector.The *dimension* variable stores the number of saved data points so far. The function has to make sure that the dimension does not exceed the size of the *record_data* array.

```
void SaveData() {
	//Save a new point in recorded_data
	//For every servo save its value
		if ( dimension < DIM ) {
			recorded_data[dimension][0] = servo_value1;
			recorded_data[dimension][1] = servo_value2;
			recorded_data[dimension][2] = servo_value3;
			recorded_data[dimension][3] = servo_value4;
			recorded_data[dimension][4] = servo_value5;
			dimension++;
		}
}
```

SetServosFromData() sets all the servos with the current position from the recorded_data. After every called to the function position is incremented. When it reaches the end of the *recorded_data* the sgn is changed from 1 to -1 and the position is now decremented until it reaches the first data point.

```
void SetServosFromData() {
	//Set all servos with the current position from recorded_data
	//Position increments to the last recorded data then it decrements
	//to the first one and so one
	if (!enable)
		return;

	//Reset sgn if position passed the bounds
	if ( position >= dimension || position < 0 ) {
		sgn *= -1;
		position += sgn;
	}

	//Write to the servos the current data point
	servo1.write(constrain(recorded_data[position][0], 0, 180));
	servo2.write(constrain(recorded_data[position][1], 0, 180));
	servo3.write(constrain(recorded_data[position][2], 0, 180));
	servo4.write(constrain(recorded_data[position][3], 0, 180));
	servo5.write(constrain(recorded_data[position][4], 0, 180));
	position += sgn;
}
```

3.4 User manual

The main control of the robot is the pot arm. The robot tries to reproduce the movement of the pot arm, so the control of the robot is made very easily. The pot arm also has a potentiometer at the end to control the position of the gripper. The control box contains the rest of the controls. There is a main power switch and 5 more switches to cut of the power of every servo, as mentioned above.

The robot has 2 more switches to control the functionalities. The right switch enables or disables the robot. To enable the robot the switch has to be in the up position, this is also indicated by a red LED. The left switch is used to select the mode of operation: the middle position is for mode 0 (the normal mode), the down position is for mode 1 (record mode), this is indicated by a yellow LED located at the bottom of the switch. The up position of the switch is for mode 2 (play mode), this is indicated by a green LED.

To record the movement of the robot the mode has to be set to 1 with the switch. To end the recording the switch has to be put in mode 0 (middle position). When selecting the record mode, all previously stored data will be lost.

To play the recorded data, the mode has to be set to 2 (up position of the switch). The robot will start to execute the commands stored in the recorded data. The robot will continue to play the recorded data until the mode is changed.

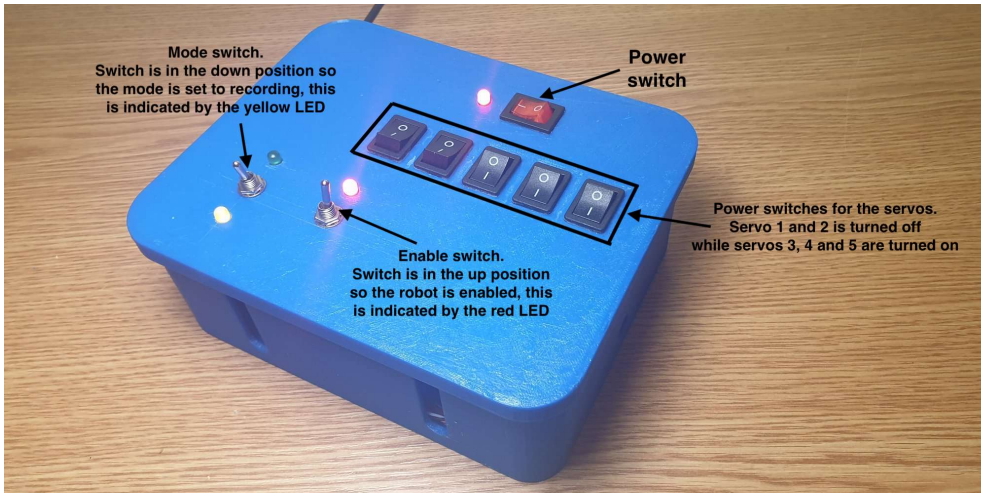

Figure 3.19 User manual of the control box

3.5 Improvements

Lots of improvements can be made to this robot. One improvement is to use stepper motors instead of servos. Although steppers require additional drivers and gear reduction, they offer a better accuracy compared with a servomotor. Another improvement is to use metal parts for the arms because they offer greater rigidity compared to a 3D printed part which is quite flexible. Lastly, the robot is an open loop system, i.e., the controller doesn't know the position of the joints. Using a closed controlled circuit and a more complex code the accuracy of the robot can be increased significantly.

CONCLUSION

The latest technological improvements from the past decade opened up new areas of knowledge to the common consumer.

This robot was made to demonstrate that the field of 3D printing, electronics, designing, programming and engineering can be combine together into one single product.

BIBLIOGRAPHY

[1]. How Do Servo Motors Work?: https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html
[2]. Arduino Mega 2560: https://store.arduino.cc/mega-2560-r3
[3]. Industrial Robot: https://en.wikipedia.org/wiki/Industrial_robot
[4]. Robots and their Arms: http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm
[5]. Buck converter: https://en.wikipedia.org/wiki/Buck_converter
[6]. Electronics Primer: How to Solder Electronic Components: https://www.sciencebuddies.org/science-fair-projects/references/how-to-solder
[7]. Soldering iron: https://en.wikipedia.org/wiki/Soldering_iron
[8]. Parts of a 3D Printer: https://3dinsider.com/3d-printer-parts/
[9]. Arduino loop(): https://www.arduino.cc/reference/en/language/structure/sketch/loop/