E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Budget inFORM, a Dynamic Shape Display

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Yuxiang Xu

Tutor: Asier Marzo

Pamplona, 9 September 2022

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Acknowledgements

I want to thank my director Asier Marzo for the guidance and help, without him I would have been really lost in the development. Also I'm really grateful for the members of UpnaLab that helped me on the smallest things and I appreciate their support.

And my thanks to everyone of my friends who encouraged me during the hardships.

Summary

The project consists in the design, assembly and programming of a budget inFORM, a Dynamic Shape Display of MIT. Each of the pixels of the display will be driven by a micro servo motor. These motors will be controlled by a PWM controller and the signals will be send through an Arduino Nano board. The approach will be modular, the minimum size will be a 2x2 module and each of them can be stacked up and scale up. All the components can either be printed with a 3d Printer or purchased cheaply.

With this device the user will be able to control the pixels and create patterns to interact with the physical world.

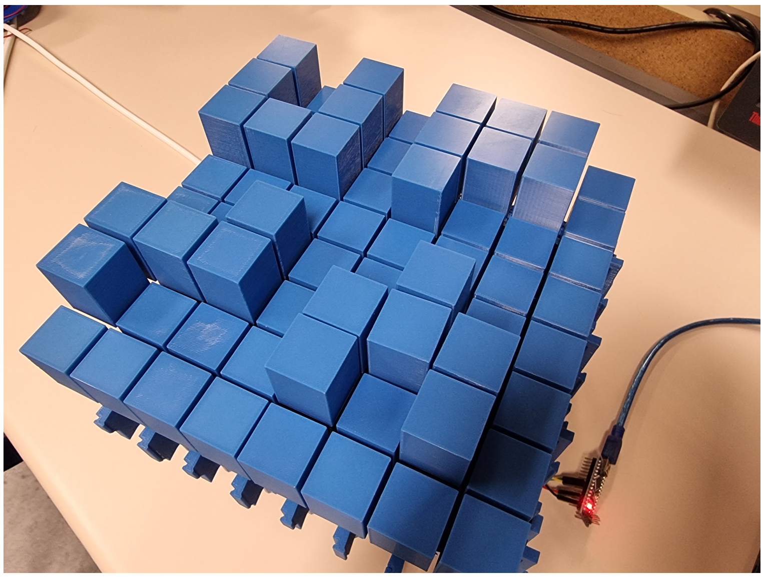


Figure 1: Budget inFORM displaying UPNa

Key words

InForm, Dynamic Shape Display, servo motor, PWM controller, Arduino, modular, Python, 3d Printer

Contents

List of Figures

List of Code Listings

1. Motivation and Objective

The main objective is to replicate the MIT inFORM in a budget friendly and simple to assemble.

In order to do that, we will be designing a modular structure where it will hold the micro servo motors. To transform the rotational movement of the servo motors into a lineal one, we will use a rack and pinion linear actuator. These components will be printed using a 3D printer.

To control the servos, we will be using a PWM (Pulse-width modulation) controller, a PCA9685 16-Channel Servo Driver, that's compatible with Arduino. We will use the library provided by Adafruit.

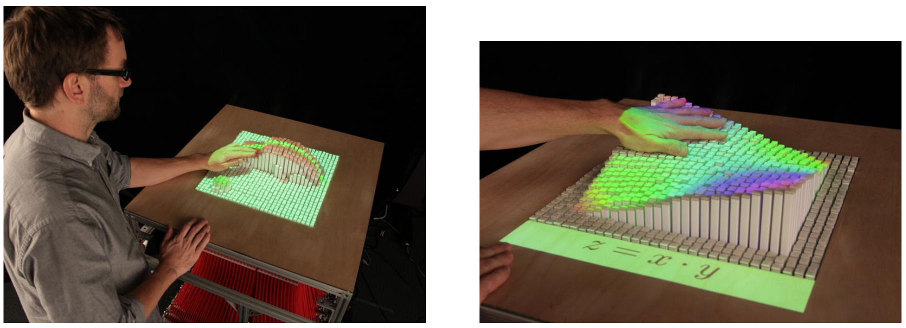
Lastly, to send the pixels to display, we will be using Python through the serial port to the Arduino board. There we can develop the applications needed for the user to control.

We are aiming to create a 8x8 board and the project can be divided into 4 sections: Mechanical, Electromechanical, Electronics and programming.

2. State of the Art

inFORM is a Dynamic Shape Display that can render 3D content physically, so users can interact with digital information in a tangible way. inFORM can also interact with the physical world around it, for example moving objects on the table's surface. Remote participants in a video conference can be displayed physically, allowing for a strong sense of presence and the ability to interact physically at a distance. [1]

The replication of all their functionalities is out of the scope of this project due to the complexity and cost. Figure 2.1.



(a) Displaying a miniature car **(b)** Displaying a math function

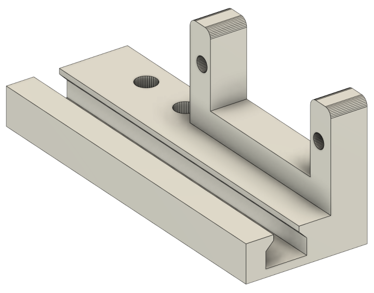
Figure 2.1: Some of their project capabilities

3. Mechanical parts

The mechanical parts consists of the modular structure, rack and pinion, top column and the legs that support the structure. All of these components will be designed using AutoCAD's Fusion 360, a 3D modeling software. In addition, they can be produced using a 3D printer.

3.1. Structure

This is the component where the servos will be installed. At a first approach, we want to design a structure to hold just one servo and from there keep iterating. Instead of coming up with the idea from scratch, we got inspired with this design from the youtuber Potent Printables in one of their videos [2] and we printed and tested a modified version from their mini version design [3]. The servo is attached using a two M2 10mm bolts and the corresponding nuts. Figure 3.1.


(a) structure in fusion 360

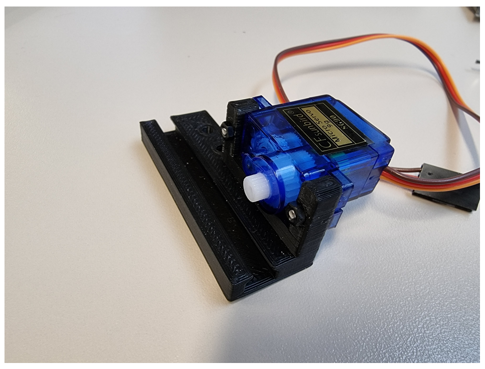
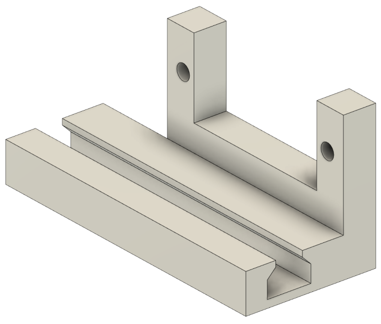

(b) structure printed with a servo attached

Figure 3.1: Modified design from Potent Printables

3.1.1. Iteration 0

As this design fits well with our servo motors, we designed the basic unit of the structure so that we can modify it and adapt to our needs. This structure is printed in the axis as shown. Figure **3.2**.


(a) structure in fusion 360

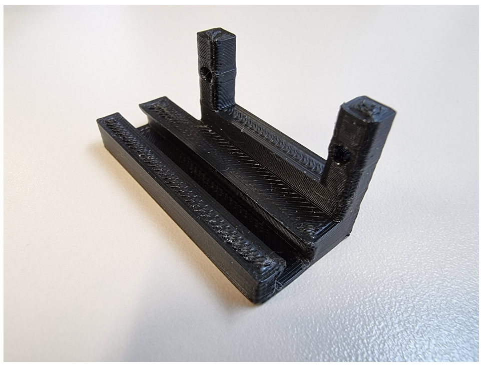
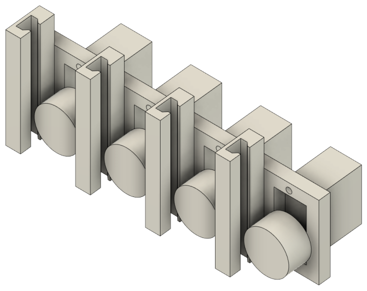

(b) structure printed

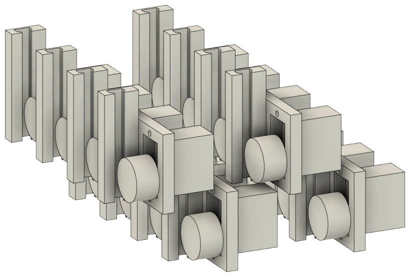
Figure 3.2: Iteration 0 of the structure

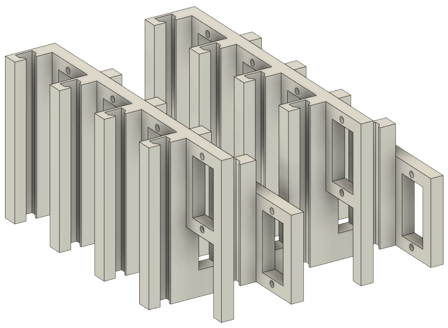
3.1.2. Iteration 1

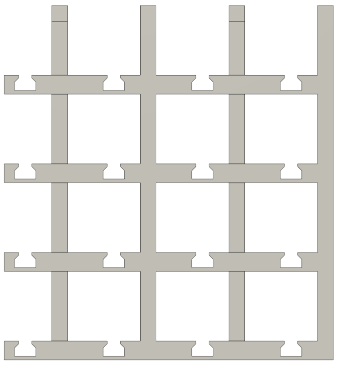
Initially we wanted to make a minimum module of 4x4 servos, as our PWM Controller can control up to 16 servos. In addition we want to make it as compact as possible so that the pixels are of a reasonable size.

Therefore we want to make a structure so that it can hold a 4x1 structure and with it try different configurations to minimize the pixel size. We managed to get a configuration of 4x4 so that the distance between each servo is 28mm, the servos are situated in to levels, a ”top” and ”bottom” layer. Figure **3.3.**


(a) structure 4x1


(b) structure 4x4 layout


(c) structure 4x4 merged


(d) structure 4x4 merged bottom view

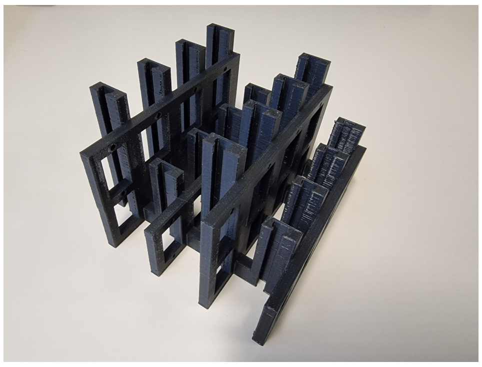

(e) structure 4x4 printed

Figure 3.3: Iteration 1 of the structure

3.1.3. Iteration 2

With the previous iteration, we realized of 2 problems, the first is that in the middle positions of the structure is not possible to place the servo in their slot due to parts of the structure impending it. From here we decided to further split the module of 4x4 into 2 modules of 2x4, this way we can assembly each part before merging them. Figure **3.6.**

The other problem is that until now we have been using Ender 3 Pro 3D printers, but these printers does not provide a good enough printing quality (figure **3.6c**), making hard to calibrate the sizes of the joints and messy first layer prints due to the elephant foot effect [4]. Fortunately, a better printer was available, the Original Prusa i3 MK3S. Figure **3.6d.**

Dovetail joint

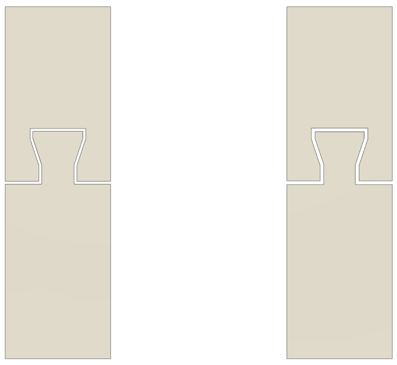
In order to merge the 2 modules of 2x4 together, a joint has to be designed, we will be using a dovetail joint. As well as the clearances needed for a good fit.

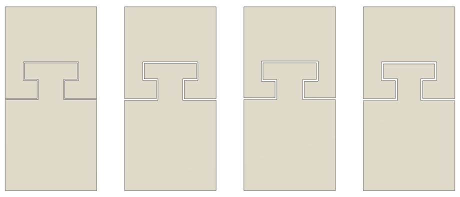
As for now, the structures are connected directly with a 6mm wide and 10mm tall union, we need to separate this union and make a joint to fit them. The first iteration we set that the maximum wide of the female joint to be less than 6mm and the for the male 4mm, with this design the printed joints were not strong enough and thus not reliable. This is why we decided to widen the joints and make modifications in the structure latter on for a reliable assembly. So in total we have three designs with different tolerances. Figure **3.4.**

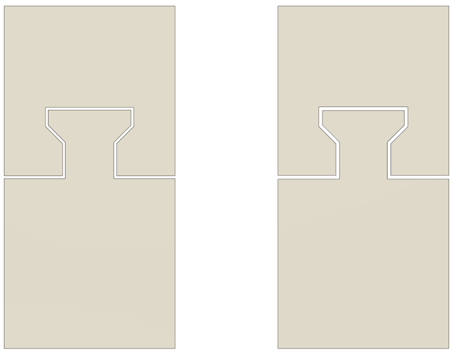
- 6mm Dovetail joint, with tolerances:
  - 0.15mm
  - 0.2mm
- 10mm Squared dovetail joint, with tolerances:
  - 0.1mm
  - 0.15mm
  - 0.175mm
  - 0.2mm
- 10mm Dovetail joint, with tolerances:
  - 0.15mm
  - 0.2mm

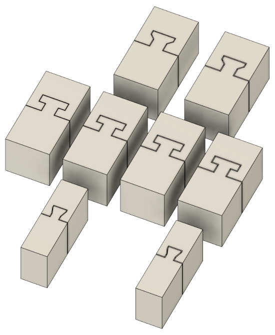
The best joint and tolerance for the Ender 3 Pro 3D printers were the 10mm Dovetail joint with 0.2mm. But the result was not satisfactory due to a bad first layer. Although the first layer can be sanded down, it is not feasible to do it with every component and would take too much time for it. Figure **3.5.**

After receiving the Original Prusa i3 MK3S printer, we proceed to print with the same dovetail joint with 0.2mm tolerance and it was too loose. This will be improved in the next iteration. Figure **3.6d.**


(a) 6mm dovetail joint


(b) 10mm Squared dovetail joint


(c) 10mm Dovetail joint


(d) All joints

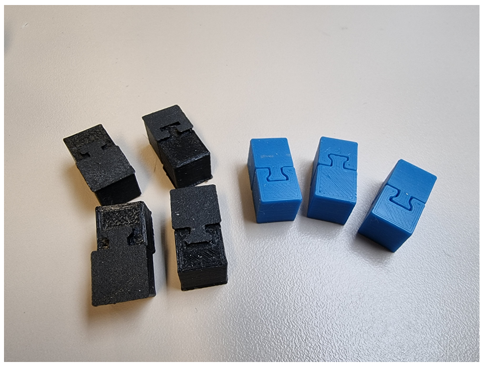

(e) Ender 3 Pro prints (black), Prusa i3 MK3S prints (blue)

Figure 3.4: Dovetail joints

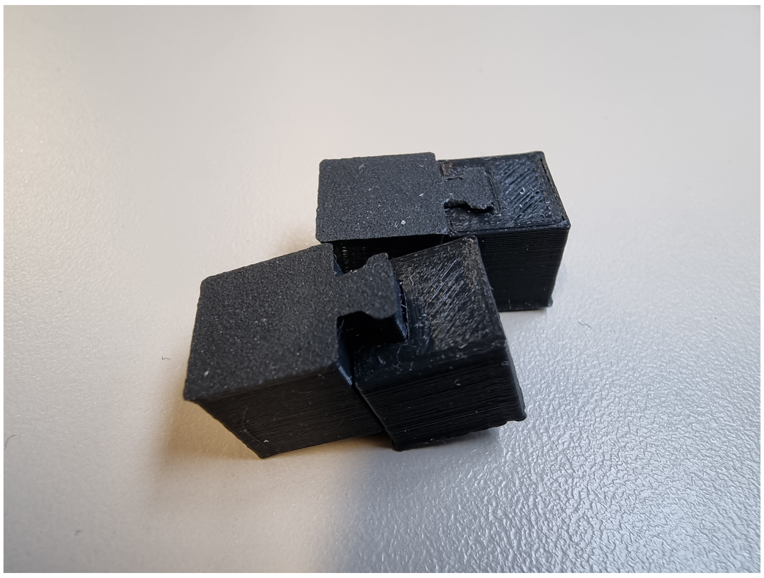
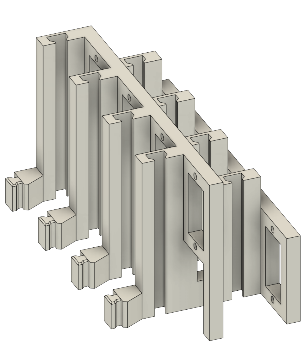
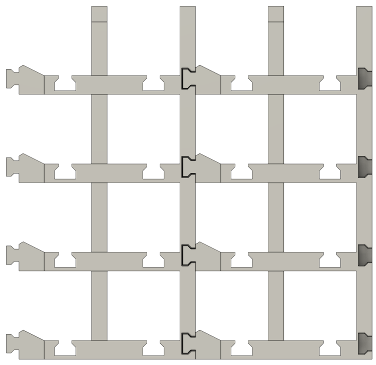


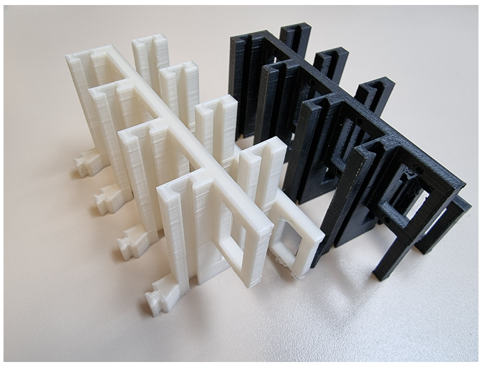
Figure 3.5: Joints Ender 3 Pro first layer problems



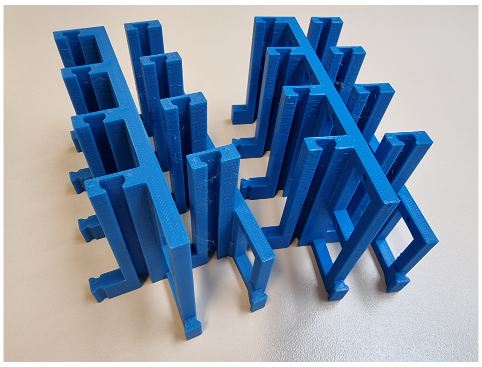
(a) structure 2x4



(b) 2 structures 2x4 bottom view



(c) structure 2x4 Ender 3 Pro



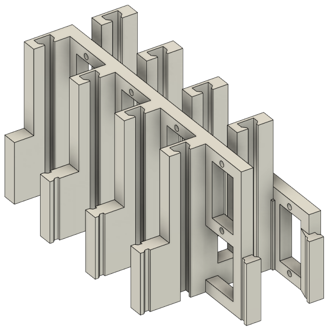
(d) structure 2x4 Prusa i3 MK3S

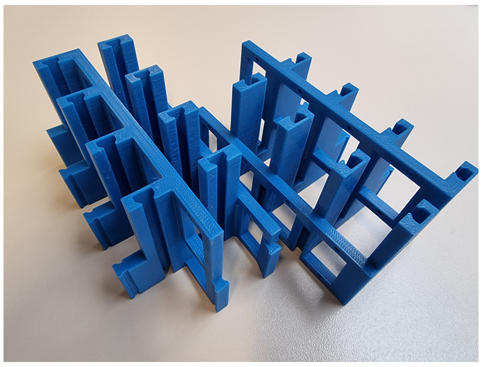
Figure 3.6: Iteration 2 of the structure

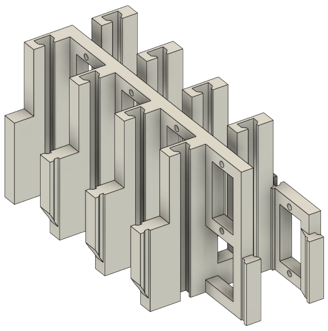
3.1.4. Iteration 3

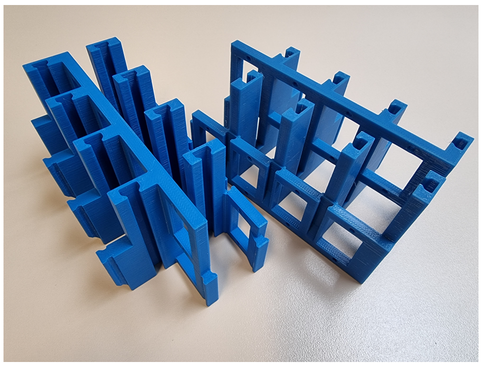
In this iteration we will be adding joints in the other axis so that the module can scale up in both of X and Y directions. In addition, with the Original Prusa i3 MK3S printers, the quality improved and the problem with elephant foot disappeared. Better tolerances can be achieved, so we deceased the tolerance to 0.1mm. This way the modules can be assembled directly without the need of glue. Furthermore, we made the joints stronger by elongating it. Figure **3.7a**.

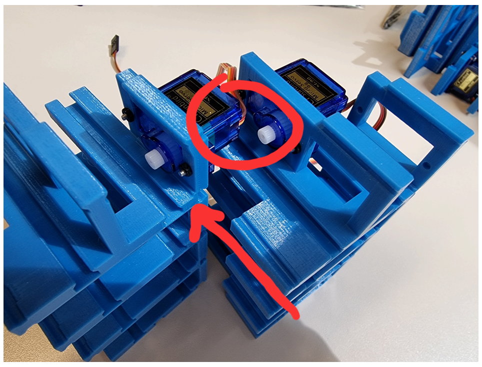
After printing it and testing it, the joints worked well. But we encountered a design fault, when the servos are mounted in the structures, it was not possible to put two modules together. The ”top” servos would hit the ”bottom” servos figure **3.7e**. Luckily, the solution was simple, and it is to invert the direction where the would be assembled. For this and to make the design 3d printing friendly, the bottom part of the joints were chamfered by 60° angle. Figure **3.7f**.


(a) structure 2x4


(b) structure 2x4 printed


(c) structure 2x4 joints inverted


(d) structure 2x4 joints inverted printed


(e) Modules hitting each other

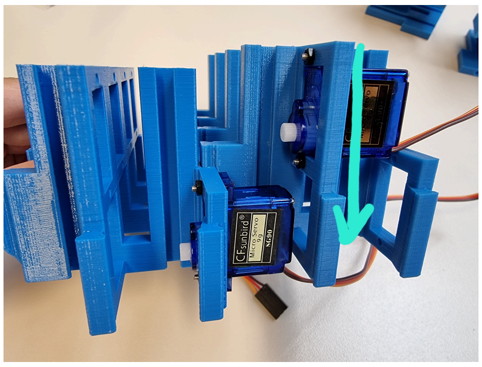
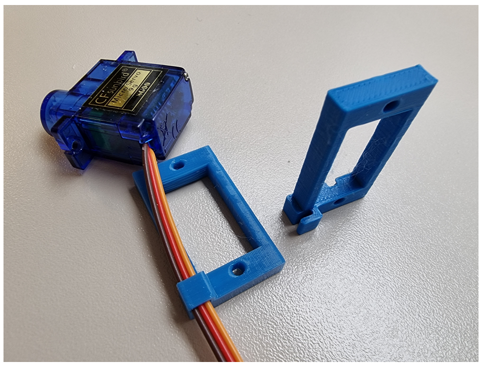

(f) Solved with the inverted joints

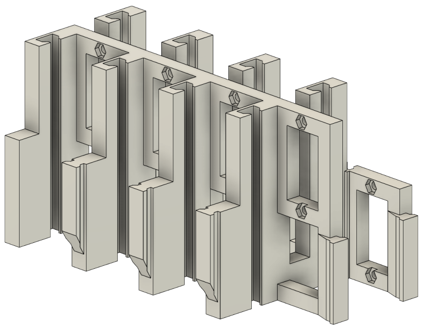
Figure 3.7: Iteration 3 of the structure

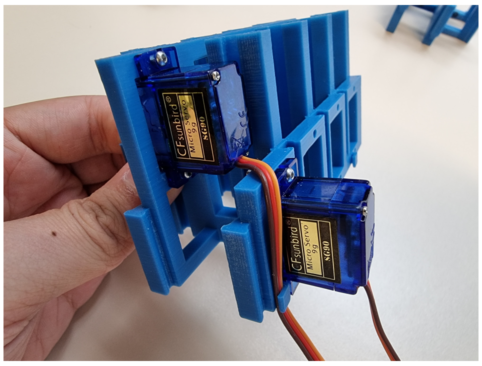
3.1.5. Iteration 4

For the last iteration minor changes were made. The first one is adding an hexagon hole for the nuts, this makes the process of assembly easier. The other change is adding a guide/hook for the cable of the servo motors to keep everything in place. Figure 3.8.

(a) Different tests to get the adequate hole size


(b) Cable holder


(c) structure 2x4

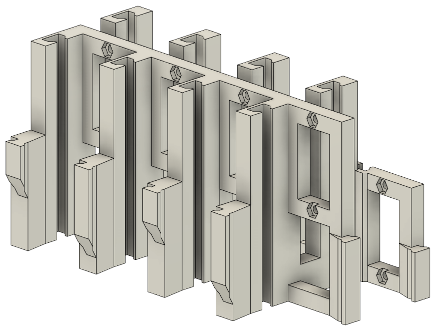


(d) structures 2x4 with 2 servos mounted

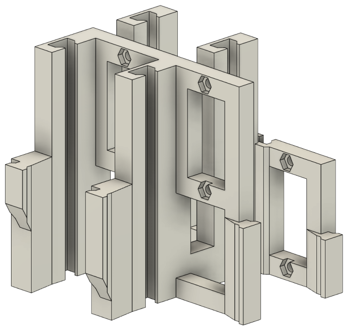
Figure 3.8: Iteration 4 of the structure

3.1.6. Final version

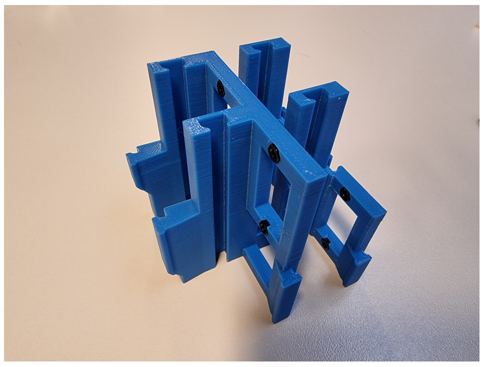
In this final iteration, we have decided to make a 2x2 module from the 2x4 module, this way it would be more versatile and more shapes can be made with the modules. The relative position of the joints were moved for better fitting. The last version of the 2x4 module is compatible with the 2x2 module. Figure 3.9.



(a) structure 2x4 compatible with 2x2



(b) structure 2x2



(c) structure 2x2 printed with nuts placed in

Figure 3.9: Final iteration of the structure

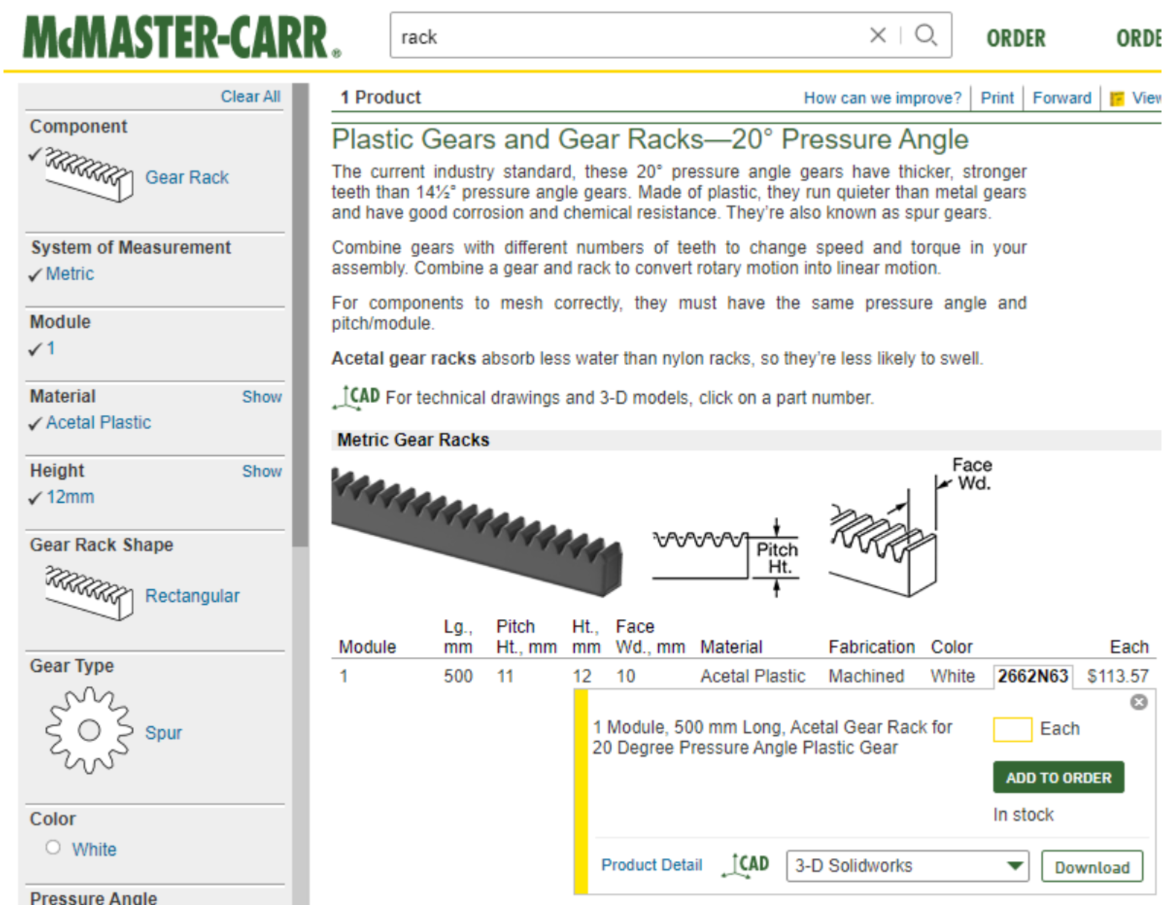
3.2. Rack and pinion

The rack and pinion linear actuator is comprised of a circular gear (the pinion) engaging a linear gear (the rack). In our case the servo will be coupled with the pinion and thus transform the rotational motion into a lineal motion of the rack.

As done in the structure section, we also took as a first approach with the design from the youtuber Potent Printables [2]. Then we replicated the design to match our needs.

3.2.1. Rack

For the rack we got a pre-made rack design from McMaster-Carr in Fusion 360. Figure **3.10**. We modified it to our needs through different try and errors. The important parameter is that is has module 1. Figure **3.11**.



Plastic Gears and Gear Racks—20° Pressure Angle

The current industry standard, these 20° pressure angle gears have thicker, stronger teeth than 14½° pressure angle gears. Made of plastic, they run quieter than metal gears and have good corrosion and chemical resistance. They're also known as spur gears.

Combine gears with different numbers of teeth to change speed and torque in your assembly. Combine a gear and rack to convert rotary motion into linear motion.

For components to mesh correctly, they must have the same pressure angle and pitch/module.

Acetal gear racks absorb less water than nylon racks, so they're less likely to swell.

CAD For technical drawings and 3-D models, click on a part number.

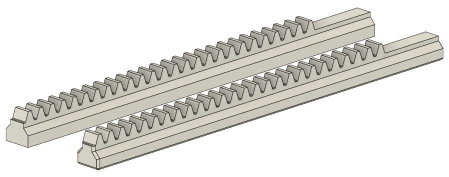
Metric Gear Racks

Module	Lg., mm	Pitch Ht., mm	Ht., mm	Face Wd., mm	Material	Fabrication	Color		Each
1	500	11	12	10	Acetal Plastic	Machined	White	2662N63	$113.57

1 Module, 500 mm Long, Acetal Gear Rack for 20 Degree Pressure Angle Plastic Gear

Each

Figure 3.10: Rack from McMaster-Carr


(a) Final versions

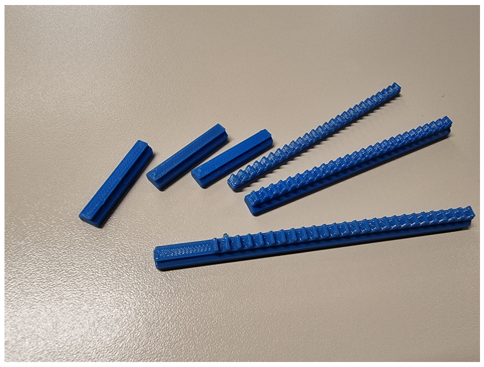
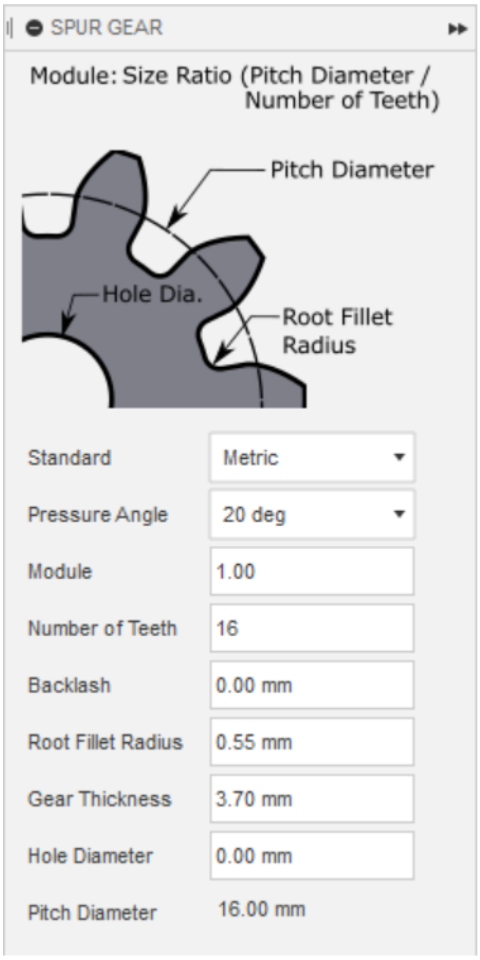

(b) Different iterations

Figure 3.11: Rack development

3.2.2. Pinion

To make the pinion, we used the SpurGear add-ins from Fusion 360 with module 1 and the following parameters in the figure **3.12a**. 16 teeth is the number of teeth on the Potent Printables designs and they are commonly used, however due to the difference between the original design and the the replicated one, there was too much play. After putting each piece together in Fusion 360 we figured out that adding 1 more tooth we would get a clearance of 0.1mm with the rack. Therefore, 17 teeth with module 1 is our chosen parameters. Figure **3.12c**.

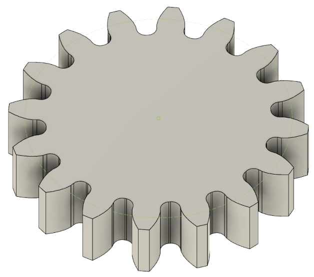
To couple the gear to the servo motor we used the provided ”blades”, the pinion has a hole to fit the blade, although the blade has to be cut, leaving only 2mm of it. This was the most secure way to assemble the gear with the servo since the 3D printers we have uses FDM technology and they are not precise enough. Figure **3.12d**.



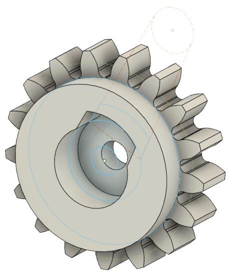
Module: Size Ratio (Pitch Diameter / Number of Teeth)

Standard	Metric
Pressure Angle	20 deg
Module	1.00
Number of Teeth	16
Backlash	0.00 mm
Root Fillet Radius	0.55 mm
Gear Thickness	3.70 mm
Hole Diameter	0.00 mm
Pitch Diameter	16.00 mm

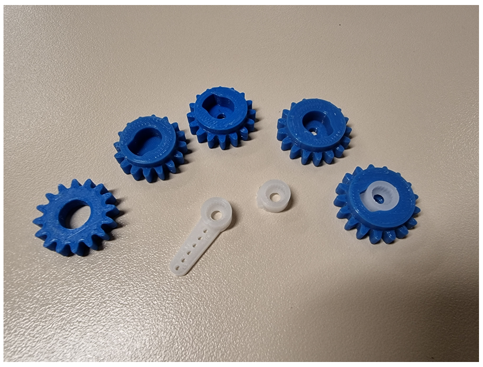
(a) parameters 16 teeth



(b) Spur gear 16 teeth



(c) Spur gear 17 teeth



(d) Iterations and example of blade

Figure 3.12: Spur gear

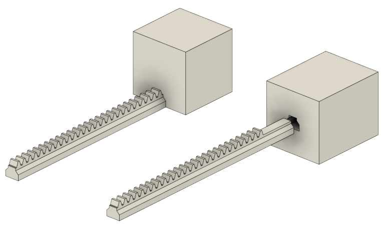
3.3. Top column

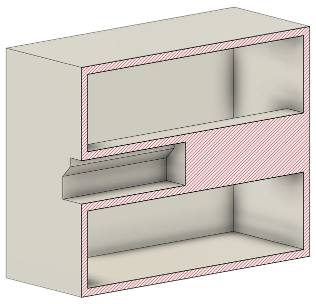
The top column is the component that's connected to the rack. It is the ”pixel” that we will be actuating. We have chosen a rectangular cuboid of dimensions of 26x26x33mm, being 33mm the height.

The first design was to print the rack with the top column attached directly, however this constrains the rack to be at the bottom of one of the sides of the top column. This is due the limitations of 3D printing, if placed in the middle it would need supports to keep the rack in place and printing in other axis was not feasible. The rack being at one side instead of centered caused undesired play of the top column.

Hence this is why we decided to separate it from the rack. Doing this we got another benefit of being able to change the top column easily to any other shape. In addition, this made the assembly process more straightforward. Figure **3.13a**.

To make the printing process faster and be more efficient with the amount of filament used, the interior of the cuboid was made hollow with the exception of the column that couples with the rack. Figure **3.13b**.


(a) Top columns in fusion 360


(b) Section view

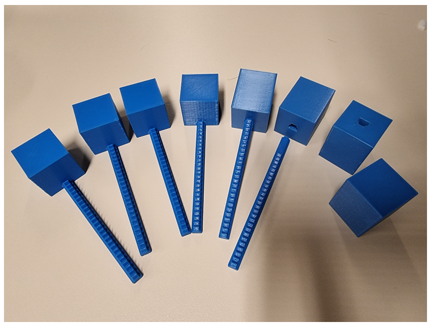

(c) Racks and top columns printed

Figure 3.13: Top column

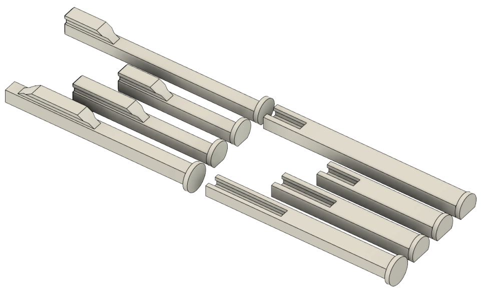
3.4. Legs

Finally to keep the structure elevated so that the rack have enough travel length and also for better cable management, some legs were designed.

3.4.1. Vertical legs

In the first place vertical legs were designed. The initial idea was to have the servos drive the racks and come out at the bottom of our structure, however for better cable management and easier printing structure we decided to invert the direction the servos will drive the racks. So lots of different legs were made to fit. The tolerances were hard to deal with as well. Figure **3.14**.

There are 2 types of legs, a ”female” and ”male” leg, since these legs makes use of the existing joints of the structure.


(a) Iterations of legs

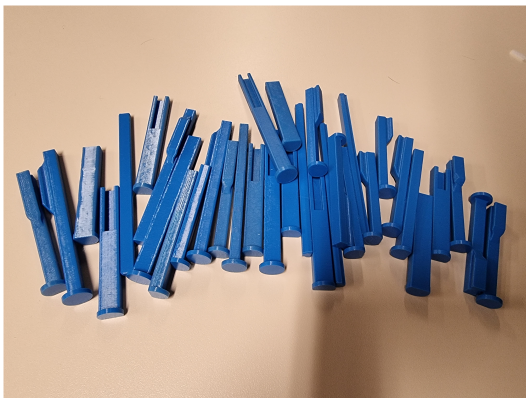
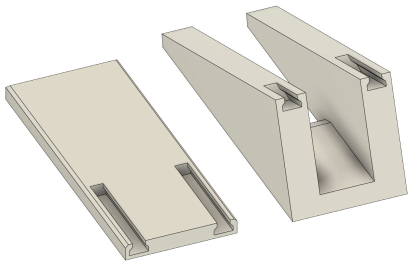

(b) All the tries of legs printed

Figure 3.14: Vertical legs

3.4.2. Horizontal legs

The horizontal legs are designed to hold the structure horizontally, the purpose is to display the structure. After printing the first version, we realized that 90° was not a good angle. So we made a another version that holds the structure at 70° angle. Figure 3.15.


(a) Legs in Fusion 360

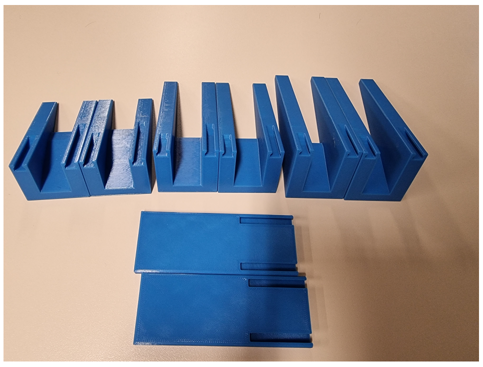

(b) All the tries of legs printed

Figure 3.15: Horizontal legs, the last pair at the right are the final version

4. Electromechanical

The main component that our system is based on are the micro servo motors figure 4.1. They are a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration [5]. This device can automatically correct its position through an error-sensing position encoder. We are using one of the simplest and cheapest servos on the market, they user a potentiometer to measure the position and the motor will move to the commanded position to minimize the error. Once they arrived to the desired position it will stop and maintain.

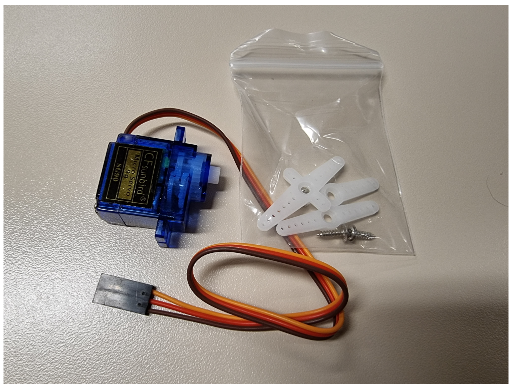


Figure 4.1: Micro servo motor with blades and screws

With our servo motor we have a rotational movement of around 180° or π radians, this combined with the pitch diameter of our pinion we can calculate the total lineal displacement we will get, as the pitch diameter is 17mm we get 26.69mm of displacement.

$$Displacement = \frac{diameter * \pi}{2}$$

The micro servos are quite responsive and can manage to go from 0° to 180° in less than 350ms, as a initial test this seems slow. It consumes at a maximum of 400mA at 5V, under slower loads it averages around 25mA. The currents are taken from observing the display of a lab power supply.

The servo motors are controller through a PWM (pulse-width modulation) signal, a series of repeating pulses of variable width where either the width of the pulse (most common modern hobby servos, our servos) or the duty cycle of a pulse train determines the position to be achieved by the servo [6].

These servos are connected through a standard three-wire connection: two wires for a DC power supply and one for control, carrying the control pulses:

- signal, yellow wire
- power, VCC, red wire
- ground, GND, black wire

The servo expects to receive a pulse every 20 ms (50Hz), by modifying the width of the pulse we can control the angle of rotation. Figure 4.2.

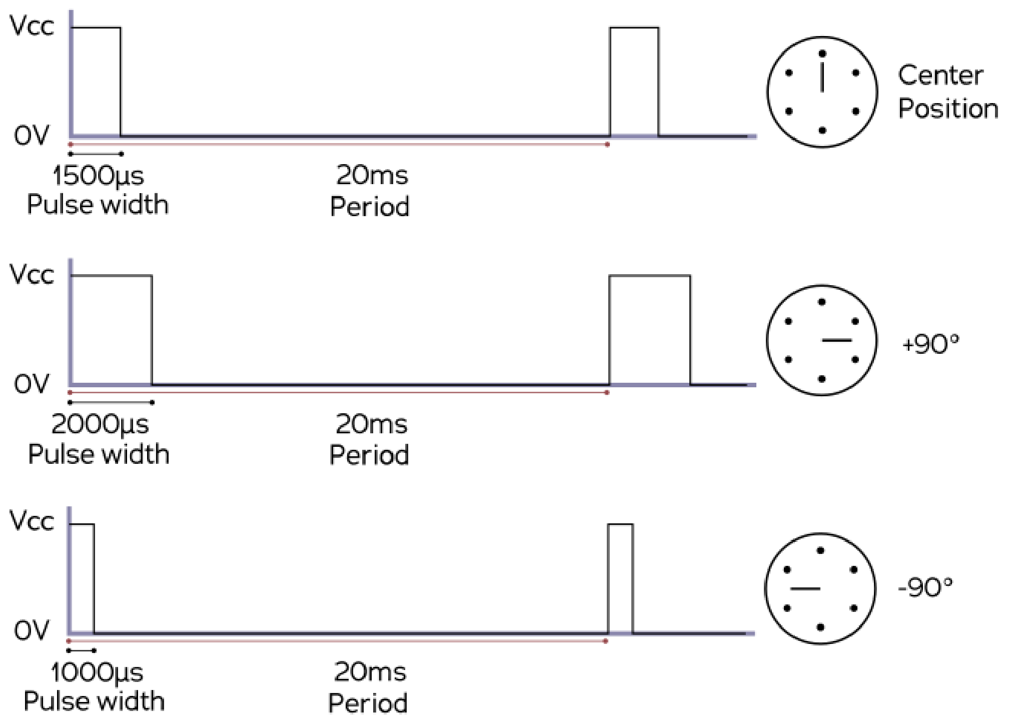


Figure 4.2: Diagram showing typical PWM timing for servo motors [6]

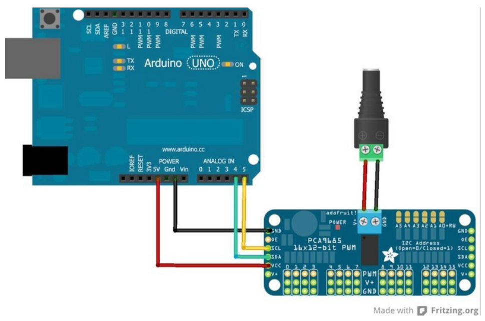
5. Electronics

We will be using an Arduino Nano as our microcontroller. We have chosen Arduino for its open-source electronics platform based on easy-to-use hardware and software, perfect for quick prototyping and affordable [7].

The arduino will receive the instructions over I2C and send them to the servos through the PWM controllers. The exact model is PCA9685 16-Channel Servo Driver, which can drive up to 16 servos over I2C with only 2 pins. This PWM controller will drive the 16 channels simultaneously with no additional Arduino processing overhead. In addition up to 62 of these controllers can be chained up to drive a total of 992 servos [8].

This is the wiring Arduino -> PWM controller figure 5.1:

- +5v -> VCC (this is power for the BREAKOUT only)
- GND -> GND
- Analog 4 -> SDA
- Analog 5 -> SCL



(a) Diagram from Adafruit [9]

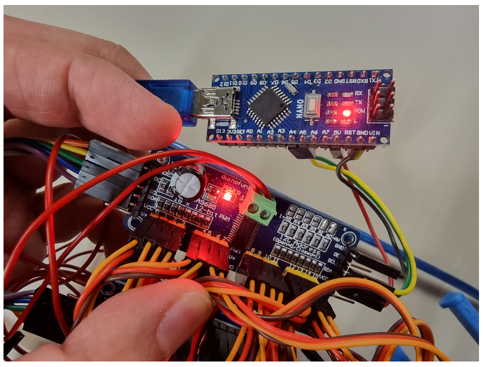

(b) Wiring

Figure 5.1: Wiring Arduino with PWM controller

Each PWM controller can be addressed with the address jumpers on the upper right edge of the board. To program the address offset we need to solder to bridge the corresponding address jumpers. The I2C base address is 0x40, each of the address jumpers works as binary, in other words by bridging the rightmost jumper we get the address 0x41 and etc. Figure 5.4.

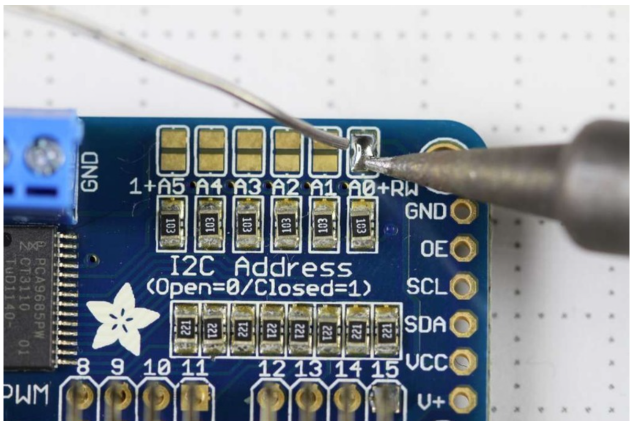
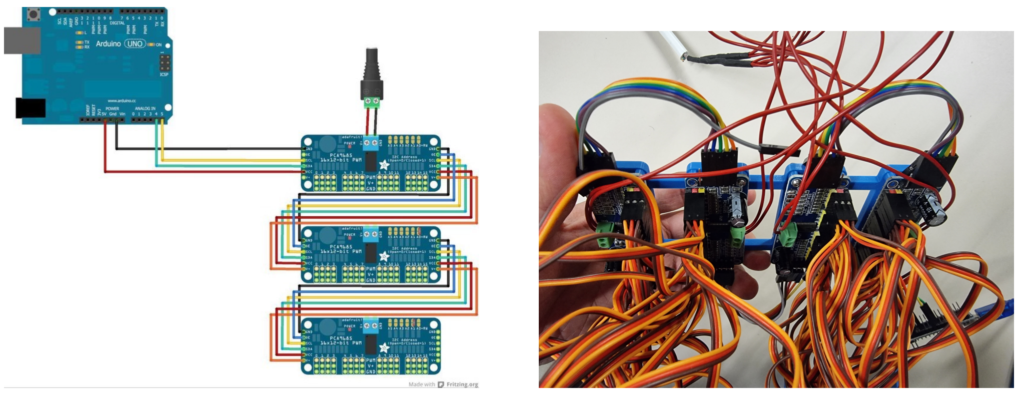


Figure 5.2: Soldering of an address jumper [10]

For chaining the PWM controllers we just need to wire them in series, for our project we decided to not connect the V+ pin. Figure 5.3.



(a) Diagram from Adafruit [10]   **(b)** Chaining

Figure 5.3: Chaining drivers

When connecting the servos to the PWM controllers we just need to match the wire's colours with the socket's colours.

For powering the servo motors an external power supply is needed, during the process of prototyping and testing we used a lab power supply at 5V and 4A max. Nevertheless, for our project of 8x8 board, we will need to power a total of 64 servos, taking into account the maximum current of 0.4A per servo, we will need a power supply that can satisfy 25.6A at 5V, even though it is not likely that it will ever require such amount of power. To be safe and not overheat our components we opted for a 30A, 5V power supply. Figure 5.4.

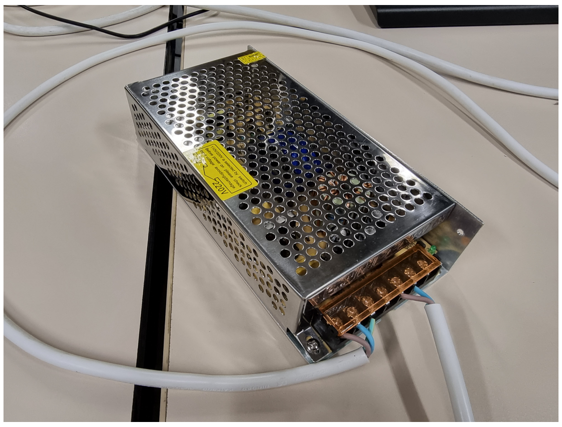


Figure 5.4: Power supply from Amazon

6. Programming

The programming was done in two languages: Arduino and Python. Arduino was needed evidently for the Arduino Nano board and their IDE was easy to use and intuitive. On the other hand Python has handy libraries such as OpenCV for computer vision. With Arduino we set up the board to receive the commands and drive the servos. With Python we will create said commands. All the code can be found in the code annex section **A**.

6.1. Connecting Python with arduino

We used the Adafruit PCA9685 library and used the their servo example file as the base for programming. The communication between Python and Arduino is done through serial, by using the Pyserial library in Python. We used Jupyter notebook for the tests. The baurate is set at 500.000Bd for faster transmission.

```
import serial
arduino = serial.Serial(port='COM3', baudrate=500000, timeout=.1)
%code
arduino.close()
```

Code listing 6.1: Stabilising serial connection in Python

One of the things we needed to solve was how to address each servo. As the PWM controllers has 16 channels from 0 to 15 and our minimum module size is 2x2, we will arrange them in groups of 4. The arrangement of 16 servos in a 4x4 module will be as such, each number corresponding the channel it is connected to.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

With this configuration we can create a recursive algorithm that takes as input a square matrix that's multiple of 2x2 and map the positions to their corresponding channel in a command line.

```
def createCommand(matrix):
    command = ""
    delimitator = ","
    n = len(matrix)
    if n%2 != 0:
        print("The matrix size is incorrect, not a multiple of 2x2"
           )
        return command

    if n == 2:
        command += str(matrix[0,0]) + delimitator
        command += str(matrix[0,1]) + delimitator
        command += str(matrix[1,0]) + delimitator
        command += str(matrix[1,1]) + delimitator
    else:
        midpoint = (n//2)
        command += createCommand(matrix[0:midpoint,0:midpoint])
        command += createCommand(matrix[0:midpoint,midpoint:])
        command += createCommand(matrix[midpoint:,0:midpoint])
        command += createCommand(matrix[midpoint:,midpoint:])

    return command
```

Code listing 6.2: Mapping matrix to command

On the Arduino side we just need to read each integer and store it in an array. Since our project is a 8x8 board, we will need to store 64 values. The first 16 values will correspond to the first PWM controller and successively with the next groups of 16 values.

```
void setServoHeight(int values[]) {
  for (byte i = 0; i < 16; i++) {
    for (int j = 0; j < N_DRIVERS; j++) {
      pwm[j].setPWM(i, 0, heightToPulselen(values[i + j * 16]));}
  }
}
```

Code listing 6.3: Function in arduino to move each servo

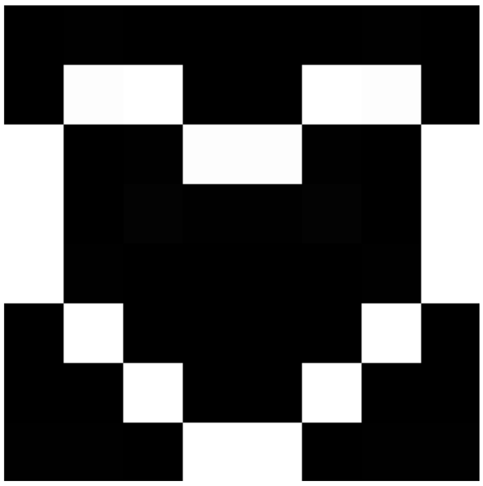
On the other hand we need to adjust the maximum displacement of each servo changing the values of pulse length count. We set the minimum value to 90 and maximum to 480 and these values do not seem to overstrain our servo motors. The caveat is that each unique servo has its own ”perfect” minimum and maximum pulse length values. This was a hard chore to calibrate each of the 64 servos, so we did not do it. When mounting the servos we leveled the 0° or ground value with the rack and pinion and not leveled the maximum value. Code **A.2**.

6.2. Applications

We developed 2 applications. All of them requires the functions and imports from code **A.4**.

6.2.1. First application

This is the first application, with this we can paint our module with a image interface. Pressing the keys [’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’] will save the files in each slot. To show the saved files press [’q’,’w’,’e’,’r’,’t’,’y’,’u’,’i’], each corresponding to the order. Code **A.5**. Figure **6.1**.


(a) Image interface

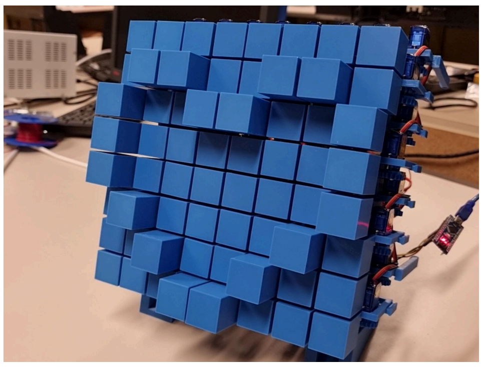

(b) Shape in the module

Figure 6.1: Heart shape

6.2.2. Second application

The second application will show predefined math function in loop. Pressing A or D key will change the functions shown. Code **A.6**. Figure **6.2**.

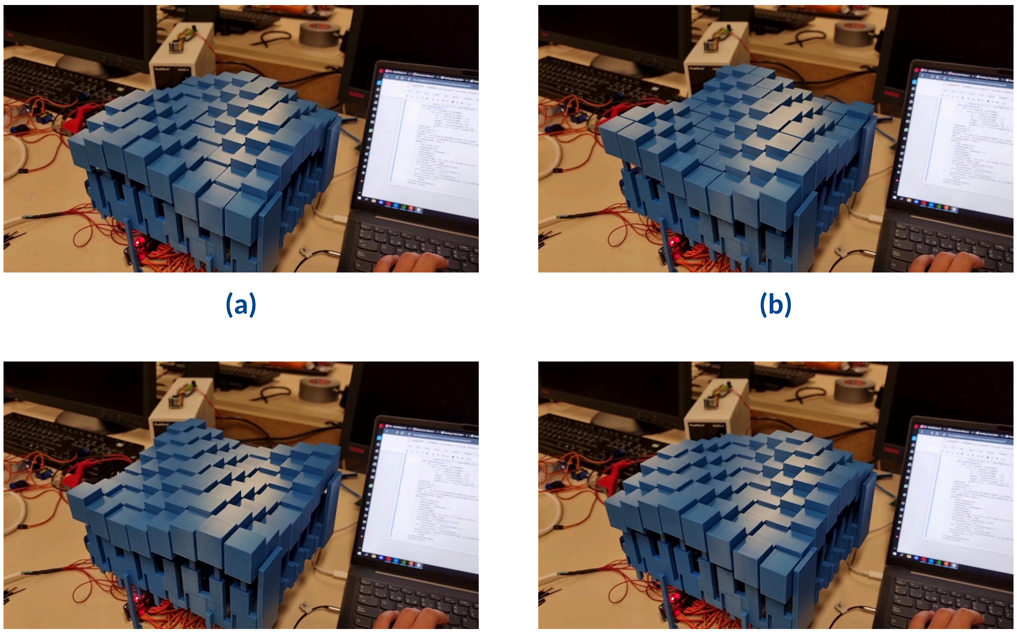


Figure 6.2: Displaying a sin wave diagonally

7. Future work

Due to the lack of time, these are some of the applications that are not done.

7.1. Kinect

The first thing we wanted to add was a kinect device, a motion sensing input device that can map depth. This is one of the main features shown in the inFORM project. This would be the most intuitive interface for our user to manipulate our device and showcase its potential. Using the kinect the user could move objects remotely.

7.2. Projector

A projector mounted on top could be used to display a map with different relief levels or any other object.

7.3. Web service

ESP32 could be used to create a web interface so the user can connect to the different applications and control our device.

7.4. More shapes

With the modular approach, we could have built any shape we wanted for other applications. For instance it is possible to make a ”belt” of the servos so it can move things around. Also as it is essentially lineal actuators, it can be used to play the piano when configures in a straight line.

7.5. Different top columns

As the top column can be changed, other shapes can take place as well as placing other types of material such a square mirror. We could attach one mirror to a group of 4 servos to create different planes to reflect light.

7.6. Improvements

And lastly, many things can be improved. Such as the noise level, the durability of the servos and the travel distance by using better servos. A better design can also be made to make the pixel more compact. The coding used for this prototyping is rushed, better and more efficient coding could be made.

A. Codes

A.1. Arduino

A.1.1. Main loop

This loop reads through serial the commands and drives the servos.

```
const int N_VALUES = 64;
int values[N_VALUES];
void loop() {

  if (Serial.available() > 1 ) {
    for (byte i = 0; i < N_VALUES; i++) {
      values[i] = Serial.parseInt();
    }
    while ( Serial.read() != '\n' ); //we skip until the end of
       line
#ifdef ENABLE_DRIVERS
    setServoHeight(values);
#endif
    for (byte i = 0; i < N_VALUES; i++) {
      Serial.print(values[i]);
      Serial.print(" ");
    }
    Serial.println();
  }
}
```

Code listing A.1: Arduino main program

A.1.2. Initialisation

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#define N_DRIVERS 4
Adafruit_PWMServoDriver pwm[N_DRIVERS] = {
  Adafruit_PWMServoDriver(0x40),
  Adafruit_PWMServoDriver(0x41),
  Adafruit_PWMServoDriver(0x42),
  Adafruit_PWMServoDriver(0x43)
};
#define ENABLE_DRIVERS 1
#define SERVOMIN  90 // This is the 'minimum' pulse length count (
    out of 4096)
#define SERVOMAX  480 // This is the 'maximum' pulse length count (
    out of 4096)
#define USMIN  600 // This is the rounded 'minimum' microsecond
    length based on the minimum pulse of 150
#define USMAX  2400 // This is the rounded 'maximum' microsecond
    length based on the maximum pulse of 600
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates
#define HEIGHTMAX 255 // Maximun height value
#define ANGLEMAX 180 // Maximun angle value
void setup() {
  Serial.begin(500000);
  Serial.setTimeout(10);
#ifdef ENABLE_DRIVERS
  for (int i = 0; i < N_DRIVERS; i++) {
    pwm[i].begin();
    pwm[i].setOscillatorFrequency(27000000);
    pwm[i].setPWMFreq(SERVO_FREQ);
  }
#endif
  delay(10);
}
```

Code listing A.2: Arduino initialisation

A.1.3. Functions

```
void setAllServoHeight(int height) {
  for (int j = 0; j < N_DRIVERS; j++) {
    for (byte i = 0; i < 16; i++) {
      pwm[j].setPWM(i, 0, heightToPulselen(height));
    }
  }
}
void setAllServoAngle(int angle) {
  for (int j = 0; j < N_DRIVERS; j++) {
    for (byte i = 0; i < 16; i++) {
      pwm[j].setPWM(i, 0, angleToPulselen(angle));
    }
  }
}
void setServoAngle(int values[]) {
  for (byte i = 0; i < 16; i++) {
    for (int j = 0; j < N_DRIVERS; j++) {
      pwm[j].setPWM(i, 0, angleToPulselen(values[i + j * 16]));
    }
  }
}
void setServoHeight(int values[]) {
  for (byte i = 0; i < 16; i++) {
    for (int j = 0; j < N_DRIVERS; j++) {
      pwm[j].setPWM(i, 0, heightToPulselen(values[i + j * 16]));
    }
  }
}
int heightToPulselen(int height) {
  int pulselen = map(height, 0, HEIGHTMAX, SERVOMAX, SERVOMIN);
  return pulselen;
}
int angleToPulselen(long angle) {
  int pulselen = map(angle, 0, ANGLEMAX, SERVOMAX, SERVOMIN);
  return pulselen;
}
```

Code listing A.3: Arduino functions

A.2. Python

A.2.1. Imports and functions for sending commands

```
import numpy as np
import cv2
import serial
import time
import math
def write(data):
    arduino.write(data.encode('utf-8'))
    arduino.flush()
def read():
    data = arduino.readline()
    data = data.decode('utf-8')[:-1]
    response = np.array(data.split())
    return response
def createCommand(matrix):
    command = ""
    delimitator = ","
    n = len(matrix)
    if n%2 != 0:
        print("The matrix size is incorrect, not a multiple of 2x2"
            )
        return command
    if n == 2:
        command += str(matrix[0,0]) + delimitator
        command += str(matrix[0,1]) + delimitator
        command += str(matrix[1,0]) + delimitator
        command += str(matrix[1,1]) + delimitator
    else:
        midpoint = (n//2)
        command += createCommand(matrix[0:midpoint,0:midpoint])
        command += createCommand(matrix[0:midpoint,midpoint:])
        command += createCommand(matrix[midpoint:,0:midpoint])
        command += createCommand(matrix[midpoint:,midpoint:])
    return command
def sendCommand(matrix):
    command = createCommand(matrix.transpose()) + "\n"
    write(command)
```

Code listing A.4: Imports and functions for sending commands

A.2.2. First application

```python
arduino = serial.Serial(port='COM3', baudrate=500000, timeout=.1)
N_COLUMNS, N_ROWS = 8,8
CELL_PX = 64

def repaintImage():
    for x in range(0, N_COLUMNS):
        for y in range(0, N_ROWS):
            sx = x * CELL_PX
            sy = y * CELL_PX
            height = (int) (heights[x,y])
            cv2.rectangle(img, (sx, sy), (sx+CELL_PX, sy+CELL_PX),
               (height, height, height), -1)

def onClick(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        column = (int)(x / CELL_PX)
        row = (int)(y / CELL_PX)
        #toggle from 0 to 255
        heights[column, row] = 255 - heights[column, row]
        repaintImage()
        sendCommand(heights)
    elif event == cv2.EVENT_RBUTTONDOWN:
        heights[:, :] = np.zeros((N_COLUMNS,N_ROWS),np.uint8)
        repaintImage()
        sendCommand(heights)

saveFiles = ['1','2','3','4','5','6','7','8']
showFiles = ['q','w','e','r','t','y','u','i']
```

Code listing A.5: First application, painting the board

```
heights = np.zeros((N_COLUMNS,N_ROWS),np.uint8)
time.sleep(0.1)
sendCommand(heights)
img = np.zeros( (N_COLUMNS*CELL_PX,N_ROWS*CELL_PX,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image',onClick)

while(1):
    cv2.imshow('image',img)

    keyPressed = cv2.waitKey(20) & 0xFF
    character = chr(keyPressed)
    if character in saveFiles:
        imagePath = './images/'+character+'.jpg'
        cv2.imwrite(imagePath, heights)
    elif character in showFiles:
        imageName = saveFiles[showFiles.index(character)]
        imagePath = './images/'+imageName+'.jpg'
        saved = cv2.imread(imagePath,0)
        heights[:, :] = saved[:, :]
        repaintImage()
        sendCommand(heights)
    elif keyPressed == 27:
        break

cv2.destroyAllWindows()
arduino.close()
```

Code listing A.5: First application, painting the board

A.2.3. Second application

This second application will show math function in loop. Press A or D key to change the functions shown.

```
arduino = serial.Serial(port='COM3', baudrate=500000, timeout=.1)
N_COLUMNS, N_ROWS = 8,8
CELL_PX = 64

def repaintImage():
    for x in range(0, N_COLUMNS):
        for y in range(0, N_ROWS):
            sx = x * CELL_PX
            sy = y * CELL_PX
            height = (int) (heights[x,y])
            cv2.rectangle(img, (sx, sy), (sx+CELL_PX, sy+CELL_PX),
               (height, height, height), -1)

#expect the range -1 to 1 for x,y. return from 0 to 1
def evalFuncAt(x,y):
    fr = timeStamp / 32
    if functionToUse == 1:
        return np.sin( fr ) * 0.5 + 0.5
    elif functionToUse == 2:
        return np.sin( fr + x*math.pi) * 0.5 + 0.5
    elif functionToUse == 3:
        return np.sin( fr + y*math.pi) * 0.5 + 0.5
    elif functionToUse == 4:
        return np.sin( fr + (x+y)*math.pi) * 0.5 + 0.5
    elif functionToUse == 5:
        dist = math.sqrt( x*x + y*y)
        return np.sin( fr + dist*math.pi) * 0.5 + 0.5

def updateHeights():
    for x in range(0, N_COLUMNS):
        for y in range(0, N_ROWS):
            nx = 2.0*x/N_COLUMNS - 1.0
            ny = 2.0*y/N_COLUMNS - 1.0
            height = evalFuncAt(nx, ny)
            height = min(max(height, 0), 1)
            heights[x,y] = (int)( height*255 )
```

Code listing A.6: Second application, showing math functions

```
timeStamp = 0
step = 2
functionToUse = 1
N_FUNCTIONS = 5
heights = np.zeros((N_COLUMNS,N_ROWS),np.uint8)
img = np.zeros( (N_COLUMNS*CELL_PX,N_ROWS*CELL_PX,3), np.uint8)
cv2.namedWindow('image')
while(1):
    time.sleep(0.03)
    updateHeights()
    timeStamp += step
    sendCommand(heights)
    repaintImage()
    cv2.putText(img, 'Esc exit, a/d change function', (0,50), cv2.
       FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 1, cv2.LINE_AA)
    cv2.imshow('image',img)
    keyPressed = cv2.waitKey(20) & 0xFF

    if keyPressed == ord('a'):
        functionToUse = functionToUse + 1
    elif keyPressed == ord('d'):
        functionToUse = functionToUse - 1
    functionToUse = min(max(functionToUse, 1), N_FUNCTIONS)
    if keyPressed == 27:
        break
cv2.destroyAllWindows()
arduino.close()
```

Code listing A.6: Second application, showing math functions

Bibliography

[1] S. Follmer, D. Leithinger, A. Olwal, A. Hogge, and H. Ishii, “inform: dynamic physical affordances and constraints through shape and object actuation.” in *Uist*, vol. 13, no. 10, 2013, pp. 2501–988.

[2] Potent Printables, “Diy linear servo actuator, 3d printed,” 2018. [Online]. Available: https://youtu.be/2vAoOYF3m8U

[3] ——, “Diy linear servo actuator, 3d printed,” 2018. [Online]. Available: https://www.thingiverse.com/thing:3170748

[4] B. Zhang, “3d printing tips: How to fix elephant foot?” 2021. [Online]. Available: https://ecoreprap.com/3d-printing-elephant-foot/

[5] Wikipedia contributors, “Servomotor — Wikipedia, the free encyclopedia,” 2022, [Online; accessed 5-September-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Servomotor&oldid=1066310615

[6] ——, “Servo control — Wikipedia, the free encyclopedia,” 2021, [Online; accessed 7-September-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Servo_control&oldid=1017828885

[7] ——, “Arduino — Wikipedia, the free encyclopedia,” 2022, [Online; accessed 6-September-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Arduino&oldid=1107764546

[8] B. Earl, “Adafruit pca9685 16-channel servo driver,” 2012. [Online]. Available: https://learn.adafruit.com/16-channel-pwm-servo-driver/overview

[9] ——, “Adafruit pca9685 16-channel servo driver, hooking it up,” 2012. [Online]. Available: https://learn.adafruit.com/16-channel-pwm-servo-driver/hooking-it-up

[10] ——, “Adafruit pca9685 16-channel servo driver, chaining drivers,” 2012. [Online]. Available: https://learn.adafruit.com/16-channel-pwm-servo-driver/chaining-drivers