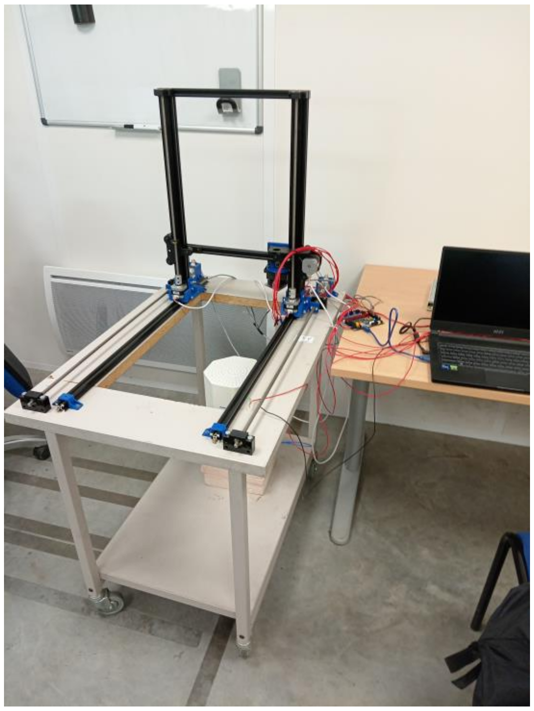
RTDetect



Marlin Firmware

Marlin is an open-source firmware that is used for 3D printers. In order to control our machine, we need to send codes in a certain format which are also known as g-codes. You can download easily the firmware. It can be downloaded from the official Marlin website (Download | Marlin Firmware (marlinfw.org)) .I highly recommend the bugfix-2.1.x version.

Machine Control

In order to control the machine we need to send g-codes in a proper format with python (we can also send gcodes using pronterface software but that does not allow us to do the automation that python give us) .

```
import serial
import time



def command(ser, str_command):
    ser.write(str.encode(str_command))
    time.sleep(1)
    while True:
        line = ser.readline()
        print(line)
        if line == b'ok\n':
            break

ser = serial.Serial('COM3', 115200)
time.sleep(2)
command(ser, "G21\r\n")
```

This Python code establishes a serial communication connection, sends a specified command to the connected device (via the **`ser`** serial object in bytes), and then continuously reads and prints lines from the serial port until the received line equals the bytes representation of "ok\n". The **`time.sleep(1)`** provides a delay to allow time for the command to be processed by the connected device (A small delay after you create the ser object is important otherwise the code won't work).

Firmware Upload

In order to modify the Malin firmware, we need to open the Marlin folder in Visual Studio Code (or in an Arduino IDE). We need to install the proper extensions (Platform IO and Auto Build Marlin) and after we finish with our changes in the firmware, we have to press the “build” button and automatically a binary file with the name”firmware.bin” will be created in a certain path

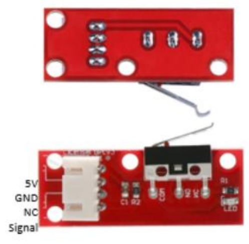
(.pio->build->LPC1768->firmware.bin). After we transfer the firmware to the SD card, we need to plug the SD card to our board, and we should see a green light. That means that the firmware has been uploaded.

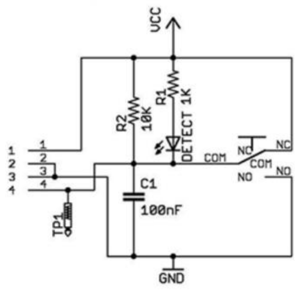
Homing

The machine needs a mechanism that would prevent it from going beyond certain limits. Hopefully Marlin allows us to do a procedure which is called homing where the machine can find where its physical boundaries are and the position of its axis. There are two types of homing: senseless homing and homing using endstops. Sensorless homing in stepper motors relies on monitoring the back electromotive force (EMF) generated when the motor is moved by an external force. By analyzing the characteristics of the back EMF, the controller can determine the motor's position, enabling homing without dedicated position sensors .Endstops on the other hand are mechanical switches which with them you can determine the reference point. Endstops are the safest way to do homing. There is a g-code in marlin which is called “g28” and is referred in the Marlin documentation as Auto-Home. What it does is that it sends a command to the machine to go to their minimum value for each axis. After it hits the endstop it will stop and it will do the same for the other axis(or if you are using sensorless homing it will go to the physical minimum of the axis and it will stop after it detect a difference in the current of the axis).After the homing finishes the machine knows where is the reference point is and it will not go beyond the physical boundaries (BED_SIZE) that you have declared in the configuration.h file in the Marlin folder of the firmware (due to software endstops).Software endstops are software mechanisms that prevent the machine to go beyond the BED_SIZE you have declared after homing was done. The BED_SIZE you have declared will be ignored before you do the homing (g28 code needs to be sent at the beginning of every movement session of the machine). Homing needs to be done every time we are using the machine for safety reasons. When you start doing homing the machine needs to be close to the reference point otherwise the g28 command will give you a timeout and you will get the response “homing failed kill () was called”.

Endstops

The job of an endstop is to detect when an axis has reached the minimum bound. A mechanical endstop is the simplest type of endstop: a simple mechanical switch positioned to trigger when the machine axis reaches the end/start of its motion. To test if the endstops were configured properly in the firmware we can test them using the g-code “M119”. The response of the g-code should refer to the mechanical endstop as “OPEN” when is not pressured and as “TRIGGERED” when it is pressured.





Endstop Configuration

1) Specify our board in the configuration.h file (you can find your board in the src-> core->boards.h file

```
#ifndef MOTHERBOARD
  #define MOTHERBOARD BOARD_BTT_SKR_V1_4
#endif
```

2) Set your serial port to –1 in the configuration.h file

```
#define SERIAL_PORT -1
```

3)

Set the min endstop state in the configuration.h file. If your endstop is a normal closed (the sig pin gives 3-5v when it is not pressed and gives 0 when it is pressed) your endstop hit state should be low

```
#define X_MIN_ENDSTOP_HIT_STATE LOW
//#define X_MAX_ENDSTOP_HIT_STATE LOW
#define Y_MIN_ENDSTOP_HIT_STATE LOW
//#define Y_MAX_ENDSTOP_HIT_STATE HIGH
#define Z_MIN_ENDSTOP_HIT_STATE LOW
```

4)
set the default_envs in platform.io your current chipset of the board you are using

```
default_envs = LPC1768
```

5)

You need to make sure that the axis will move at their minimum while homing

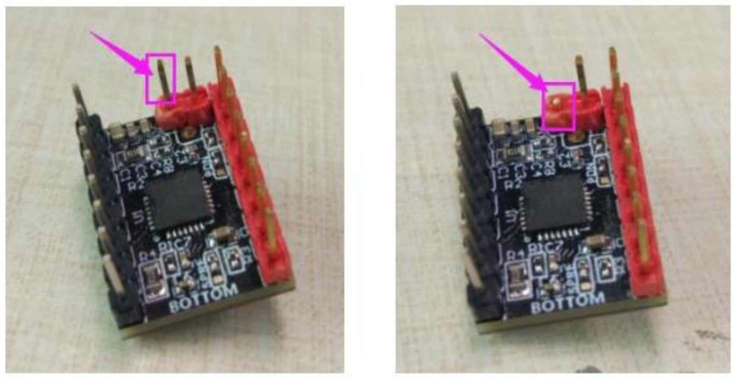
```
// Direction of endstops when homing; 1=MAX, -1=MIN
//: [-1,1]
#define X_HOME_DIR -1
#define Y_HOME_DIR -1
#define Z_HOME_DIR -1
```

For more information about configuration and wiring I recommend the following youtube channels:((314) Ed's 3d Tech - YouTube),(Welcome To My Basement - 2021 Channel Intro - Chris's Basement (youtube.com))

DIAG pin bug

Note: When using the TMC2209, if you don' t use the stallguard function, you need to cut off the DIAG pin on the driver so that the mechanical switch can work normally. The operation method is as shown below:

① Before cutting↓ ②After cutting↓



So as written in the manual if you want to use endstop switch for homing to cut this pin instead of sensorless homing it is important to cut the DIAG pin of the driver as it shown in the picture so that the limit switch can work properly

UNITS

We can set units to mm with “g21” and to inches with “g20”. Also, in order for the belt system to align with the thread system some exact configurations need to be done and the step per unit needs to be changed so that we have a certain accuracy but i did not have time for these type of settings

Objective

During my internship I was asked to find a way to for the machine to have an axis reference point and prevent the machine from going out of bounds (So I did the endstops configuration and also some 3D printing for the endstop holder) and also write a script so that the machine was going to do a cuboid movement with a certain step size for each axis and add an extra motor at the top of the machine to do the rotation of the tag

Thinking Process

So, for the machine to do a cuboid movement, first I was going to make the machine do a rectangle movement in the yz axis and if I added multiple rectangles, I would get the cuboid. I had only 3 sockets for 3 axis and two sockets for two extruders in the board so i was thinking of sending commands like “`M302 S0`” and “`M302 P1`” to make the extra motor work

Extra Motor

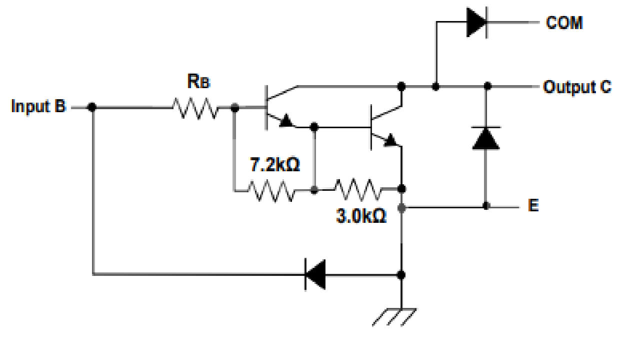
The board that I was using (bigtreetech skr pro v1.4) had only sockets for the 3 axis (x, y, z) and 2 sockets for the extruders (E0, E1). In 3D printing, an extruder is a critical component of the 3D printer responsible for feeding and melting the filament material, and then depositing it layer by layer to create the 3D object. The two key parts of the extruder are the “hotend” which is the part where the filament is melted and the motor which is responsible for driving the gears or other mechanisms that push the filament into the extruder assembly. So, I thought of using E0 for the extra motor and I sent the “M302 S0” command to allow extrusion at any temperature, the “G92 E0” command to set the current position of the extruder and finally the “G1 X10 E20 F45” command so that I can do a movement while using the extra motor.

```
command (ser,"M302 S0\r\n")
command(ser,"G92 E0\r\n")
command(ser,"G1 X10 E20 F45\r\n")
```

Unfortunately, the machine did not move, and the axis held the torque. After a few changes in the firmware, I was not able to make it work so instead I thought of using another stepper motor that would be controlled by an Arduino.

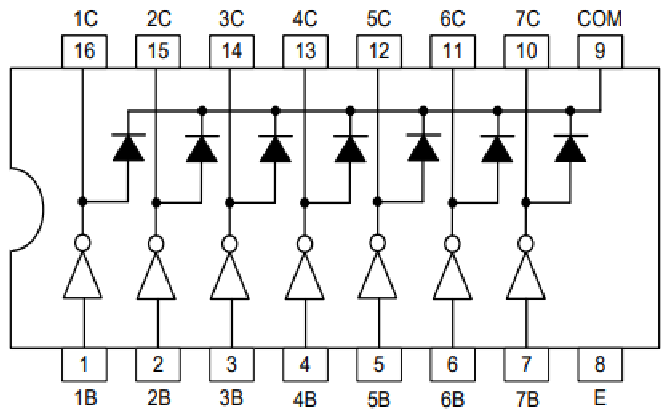
ULN2004A

The ULN2004A is a high-voltage, high-current Darlington Array containing seven open collector Darlington pairs with common emitters. Each channel is rated at 500 mA and can withstand peak currents of 600 mA and we can use it as a driver for our stepper motor.

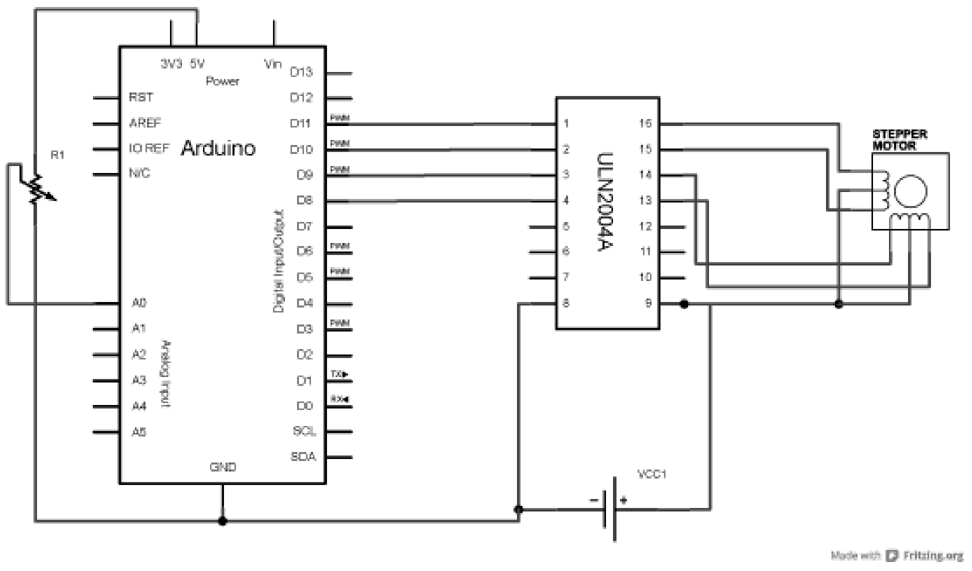


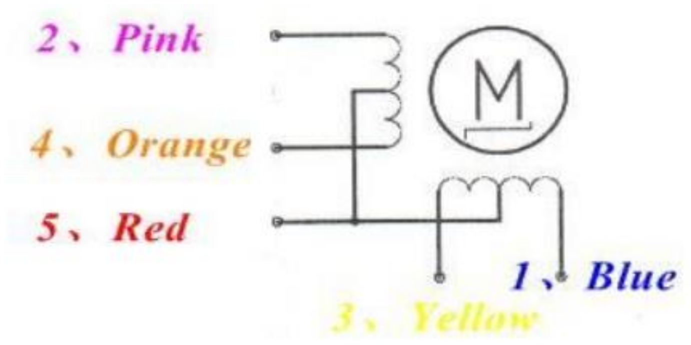
ULN2003A:	R_B = 2.7k
ULN2004A:	R_B = 10.5k

ULN2003A/ULN2004A


1C
2C
3C
4C
5C
6C
7C
COM
16
15
14
13
12
11
10
9
1
2
3
4
5
6
7
8
1B
2B
3B
4B
5B
6B
7B
E

Wiring


3V3
5V
Vin
Power
RST
AREF
IO REF
N/C
Arduino
R1
A0
A1
A2
A3
A4
A5
Analog Input
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
SCL
SDA
PWM
TX▸
RX◂
Digital Input/Output
ULN2004A
STEPPER MOTOR
VCC1
Made with Fritzing.org



In order to control the motor, we need to use an external power supply and the ULN2004A integrated circuit. The wiring needs to be done exactly how it is in the schematic with the only difference that instead of me using the pins 8,9,10,11, I used the pins 9,10,11,12(Blue in pin 16, Pink on pin 15, Yellow on pin 14, Orange on pin13)

Arduino code

```
#include <Stepper.h>
const unsigned int MAX_MESSAGE_LENGTH = 12;
static char message [MAX_MESSAGE_LENGTH];
static unsigned int message_pos = 0;
int angle = 0;
const int stepsPerRevolution = 2048;  // change this to fit the number of steps
per revolution
float float_steps = 0;
int int_steps = 0;
// for your motor

// initialize the stepper library on pins 8 through 11:
#include <Stepper.h>
Stepper myStepper(stepsPerRevolution, 9, 10, 11, 12);

int stepCount = 0; // number of steps the motor has taken
```

```
void setup () {
  Serial.begin(9600);
  myStepper.setSpeed(10);
  delay (1000);
  pinMode(7, OUTPUT);
}

void loop () {

while (Serial.available() > 0)
 {

  char inByte = Serial.read();

   //Message coming in (check not terminating character) and guard for over
message size
   if (inByte != '\n' && (message_pos < MAX_MESSAGE_LENGTH - 1) && inByte!='%')
   {
     //Add the incoming byte to our message
     message[message_pos] = inByte;
     message_pos++;
   }
   //Full message received...
   else{
    message[message_pos] = '\0';
    angle  = atoi(message);
    message_pos = 0;
    if (angle! =1234) {
    float_steps = ((float)angle*6100)/((float)360);
    int_steps = (int)(float_steps);
    Serial.print(int_steps);
    Serial.print("\n");
    myStepper.step(int_steps);
    delay (1000);
    }
    else {
      digitalWrite(7,HIGH);
      delay(5000);
      digitalWrite(7,LOW);
    }
   }
  }
 }
```

This code is designed to receive the angle via Serial communication with python and control the stepper motor accordingly. Arduino reads data from the buffers, and it saves it at the char array "message". Once it receives the character '%' (we use it as a delimiter) it will stop reading data from the buffers, it will put a '\0' at the end of the char array and using the atoi function it will get the integer value of the char array which is our angle . The led which is connected to pin 7 is activated when the angle command is 1234.I observed that the exact number of steps that the motor need in order to do a full 360 rotation is 6100 so I do a normalization (`float_steps = ((float)angle*6100)/((float)360) to find the exact number of steps you need for the angle and then you need to do a type cast from float to int  to send it to the motor)`

Python Code

```
import serial
import time

def send_to_arduino(ser,value):ser.write(str.encode(value + "%"))
time.sleep(0.1)def command(ser, str_command):
ser.write(str.encode(str_command))
time.sleep(1)
    while True:
        line = ser.readline()
        print(line)
        if line == b'ok\n':
            break
def draw_yz_cube_surface(Vy,Vz,Oy,Oz,step_y,step_z,delay): #dimensions
of the volume and the object and the increments of  y z
    dz = Vz-Oz   #distance that we need to cover
    dy = Vy - Oy
    step_num_y = int(dy/step_y)
    step_num_z = int(dz/step_z)
    y = 0
    z = 0
    if step_num_y!=1:
        for i in range(step_num_y // 2):
            for j in range(step_num_z):
                z += step_z
                command(ser, "G0 Z" + str(z) + "\r\n")
                time.sleep(delay)
            y += step_y
            command(ser, "G0 Y" + str(y) + "\r\n")
            time.sleep(delay)
            print("y increased")
            for k in range(step_num_z):
                z -= step_z
                command(ser, "G0 Z" + str(z) + "\r\n")
                time.sleep(delay)
            y += step_y
            command(ser, "G0 Y" + str(y) + "\r\n")
            time.sleep(delay)
            print("y increased")
            # set yz to zero
        print("finished iterations")
        if step_num_y % 2 == 1:
            for i in range(step_num_z):
                z+=step_z
                command(ser, "G0 Z" + str(z) + "\r\n")
                time.sleep(delay)
            y += step_y
            command(ser, "G0 Y" + str(y) + "\r\n")
            time.sleep(delay)
            #return zero
            z-=dz
            y-=dy
            command(ser, "G0 Z" + str(z) + "\r\n")
            command(ser, "G0 Y" + str(y) + "\r\n")
        elif step_num_y % 2 == 0:
            #return to zero
            y-=dy
            command(ser, "G0 Y" + str(y) + "\r\n")
    else:
        for i in range(step_num_z):
            z+=step_z
            command(ser, "G0 Z" + str(z) + "\r\n")
            time.sleep(delay)
        y += step_y
        command(ser, "G0 Y" + str(y) + "\r\n")
        time.sleep(delay)
        #return to zero
        y -= step_y
        z -= dz
        command(ser, "G0 Z" + str(z) + "\r\n")
        command(ser, "G0 Y" + str(y) + "\r\n")

def draw_cube_limits(Vx, Vy, Vz, Ox, Oy, Oz):
 x = 0
 y = 0
 z = 0

 dz = Vz - Oz
 dy = Vy - Oy
 dx = Vx -Ox

 z += dz
command(ser, "G0 Z" + str(z) + "\r\n")
 x += dx
 command(ser, "G0 X" + str(x) + "\r\n")
 y += dy
 command(ser, "G0 Y" + str(y) + "\r\n")
 x -= dx
 command(ser, "G0 X" + str(x) + "\r\n")
 y -= dy
 command(ser, "G0 Y" + str(y) + "\r\n")
 z-=dz
 command(ser, "G0 Z" + str(z) + "\r\n")

def draw_cube(Vx, Vy, Vz, Ox, Oy, Oz,step_x, step_y,step_z,delay):
    x = 0
    dx = Vx-Ox
    step_num_x = int(dx/step_x)
    for x in range(step_num_x):
        x += step_x
        draw_yz_cube_surface(Vy, Vz, Oy, Oz, step_y,step_z,delay)
        command(ser,"G0 X"+str(x)+"\r\n")
        time.sleep(delay)
    #return to zero
    x-=dx
    command(ser, "G0 X" + str(x) + "\r\n")


ser = serial.Serial('COM8', 115200)
arduino = serial.Serial(port='COM4', baudrate = 9600)
time.sleep(2)
degrees = 180
```

```
rotating_delay = (degrees * 10) / 360
send_to_arduino(arduino,str(180))
time.sleep(rotating_delay)
draw_cube(50,65,65,20,5,10,10,10,10,5)
```

First of all, after we initialize our connection to the machine and to Arduino to a different com port, we send the command to the Arduino turn 180 degrees. We calculate the delay that we need (approximately the motor needs 10 seconds to do 180 degrees). After that I send a command to draw continuous rectangle surfaces in the yz axis, while also increasing the x. In the input Vx,VY,VZ we specify the dimensions of the small cuboid (this code can be used in case we have a box with a tag inside and we want to do measuring and displacement, if we want, we can set those values to zero) and Ox,Oy,Oz are the dimension of the large cuboid volume that the machine is going to move inside. The draw_cube_limits()   function goes at the boundaries of the cube to make sure that we have set the limits correctly

Demo

In the end i was asked to do a Demo where the machine would do a cuboid movement, but it would pass only from 4 points. So first, the machine would pass from all the four points and then it would ask the user if he wants to continue (to ensure that the points that the user gave were correct. After the machine would pass again from all the points and it would stop at certain point and a green led would blink which simulates the reader and that the measure was done successfully. After this the stepper motor with the tag would turn 180 degrees and then pass again from each point and the led would blink after it passes each point.

```
import serial
import time
def send_to_arduino(ser,value):
ser.write(str.encode(value + "%"))
time.sleep(0.1)

def btt_command(ser, str_command):
```

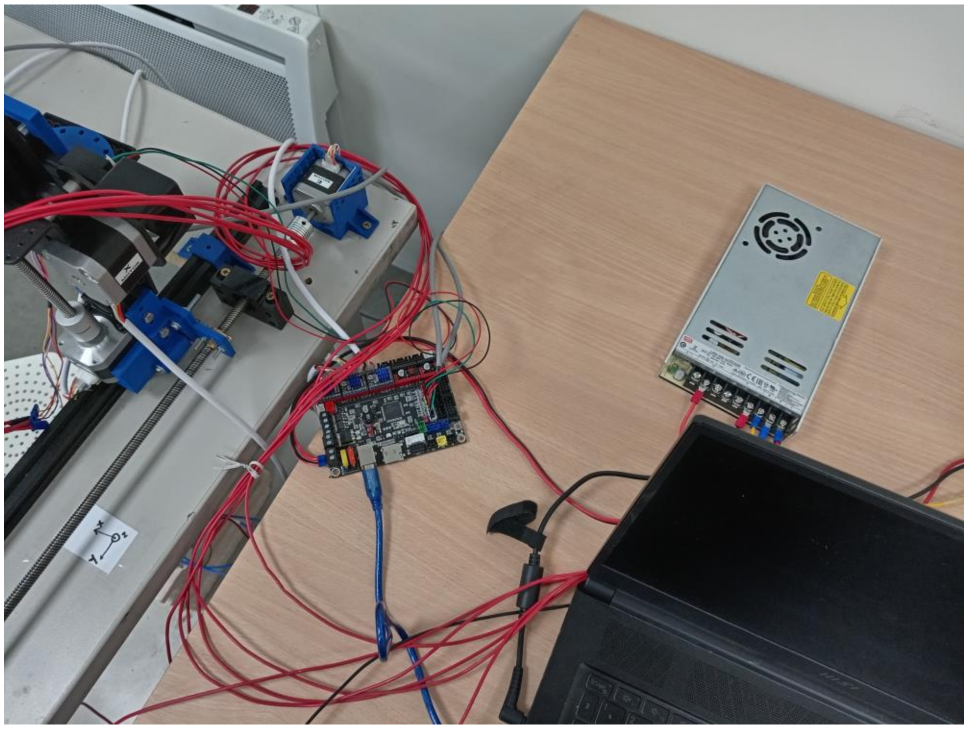
```
ser.write(str.encode(str_command))
time.sleep(1)
   while True:
        line = ser.readline()
        print(line)
        if line == b'ok\n':
            break

def draw_cube_limits(ser,dx, dy,
dz,pos_delay_x,pos_delay_y,pos_delay_z):
 x = 0
 y = 0
 z = 0

 z += dz

 btt_command(ser, "G0 Z" + str(z) + "\r\n")
 time.sleep(pos_delay_z)


draw_horizontal_surface_without_measuring_delay(btt,dx,dy,pos_delay_x,
pos_delay_y)

 z-=dz

 btt_command(ser, "G0 Z" + str(z) + "\r\n")
 time.sleep(pos_delay_z)


draw_horizontal_surface_without_measuring_delay(btt,dx,dy,pos_delay_x,
pos_delay_y)


def
draw_horizontal_surface_without_measuring_delay(ser,dx,dy,pos_delay_x,
pos_delay_y):
    x = 0
    y = 0
    z = 0
    x += dx
    btt_command(ser, "G0 X" + str(x) + "\r\n")
    time.sleep(pos_delay_x)
    y += dy
    btt_command(ser, "G0 Y" + str(y) + "\r\n")
    time.sleep(pos_delay_y)
    x -= dx
    btt_command(ser, "G0 X" + str(x) + "\r\n")
    time.sleep(pos_delay_x)
    y -= dy
    btt_command(ser, "G0 Y" + str(y) + "\r\n")
    time.sleep(pos_delay_y)

def
draw_horizontal_surface_measuring(btt,arduino,dx,dy,measuring_delay,po
s_delay_x,pos_delay_y):
    x = 0
    y = 0
    z = 0
    x += dx
    btt_command(btt, "G0 X" + str(x) + "\r\n")
    time.sleep(pos_delay_x)
    send_to_arduino(arduino, str(1234))
    time.sleep(measuring_delay)

    y += dy
    btt_command(btt, "G0 Y" + str(y) + "\r\n")
    time.sleep(pos_delay_y)
    send_to_arduino(arduino, str(1234))
    time.sleep(measuring_delay)

    x -= dx
    btt_command(btt, "G0 X" + str(x) + "\r\n")
    time.sleep(pos_delay_x)
    send_to_arduino(arduino, str(1234))
    time.sleep(measuring_delay)

    y -= dy
    btt_command(btt, "G0 Y" + str(y) + "\r\n")
    time.sleep(pos_delay_y)
    send_to_arduino(arduino, str(1234))
    time.sleep(measuring_delay)

def draw_cube(btt,arduino,dx,
dy,dz,measuring_delay,pos_delay_x,pos_delay_y,pos_delay_z):
     x = 0
     y = 0
     z = 0

     z += dz
     btt_command(btt, "G0 Z" + str(z) + "\r\n")
     time.sleep(pos_delay_z)
     send_to_arduino(arduino,str(1234))
     time.sleep(measuring_delay)

     draw_horizontal_surface_measuring(btt,arduino,dx,
dy,measuring_delay,pos_delay_x,pos_delay_y)

     z -= dz
     btt_command(btt, "G0 Z" + str(z) + "\r\n")
     time.sleep(pos_delay_z)
     send_to_arduino(arduino, str(1234))
     time.sleep(measuring_delay)

     draw_horizontal_surface_measuring(btt,arduino,dx,
dy,measuring_delay,pos_delay_x,pos_delay_y)


btt = serial.Serial('COM8', 115200)
arduino = serial.Serial(port='COM4', baudrate = 9600)
time.sleep(2)
"""
#btt_command(btt, "G28" + "\r\n")
#btt_command(btt, "G21" + "\r\n") regulate to millimeters

#calculating rotating delay
degrees = 180
rotating_delay = (degrees * 10) / 360
draw_cube(btt,arduino,150,850,150,5)
#rotating 180 degrees
send_to_arduino(arduino,180)
time.sleep(rotating_delay)
draw_cube(btt,arduino,150,850,150)
"""
#btt_command(btt, "G28 X Y Z" + "\r\n")
#btt_command(btt, "M119" + "\r\n")
```

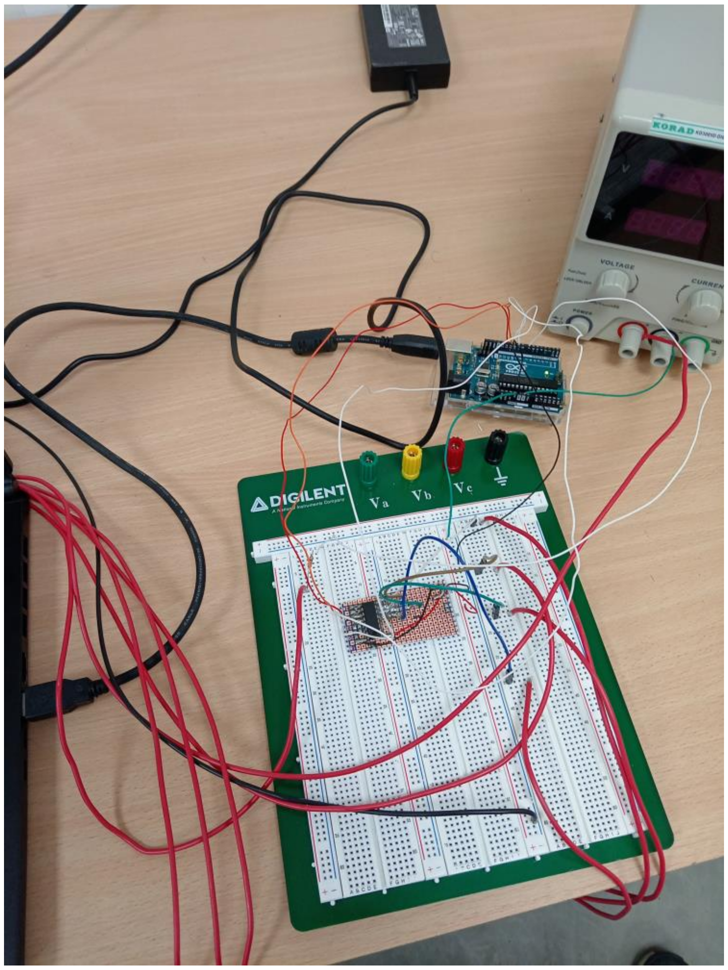
```
#btt_command(btt,"G0 Y0\r\n")

btt_command(btt, "G28 X Y Z" + "\r\n")
draw_cube_limits(btt,110, 320, 25,2,5,5)
choice = input("Do you want to continue?:[y/n]")
while True:
    if choice =='y':
        draw_cube(btt,arduino,110,320,25,3,2,5,5)
        degrees = 180
        rotating_delay = (degrees * 10) / 360
        rotating_delay+=3
        send_to_arduino(arduino,str(180))
        time.sleep(rotating_delay)
        draw_cube(btt,arduino,110,320,25,3,2,5,5)
        break
    elif choice == 'n':
        exit(0)
    else:
        print("Wrong Input!")
```

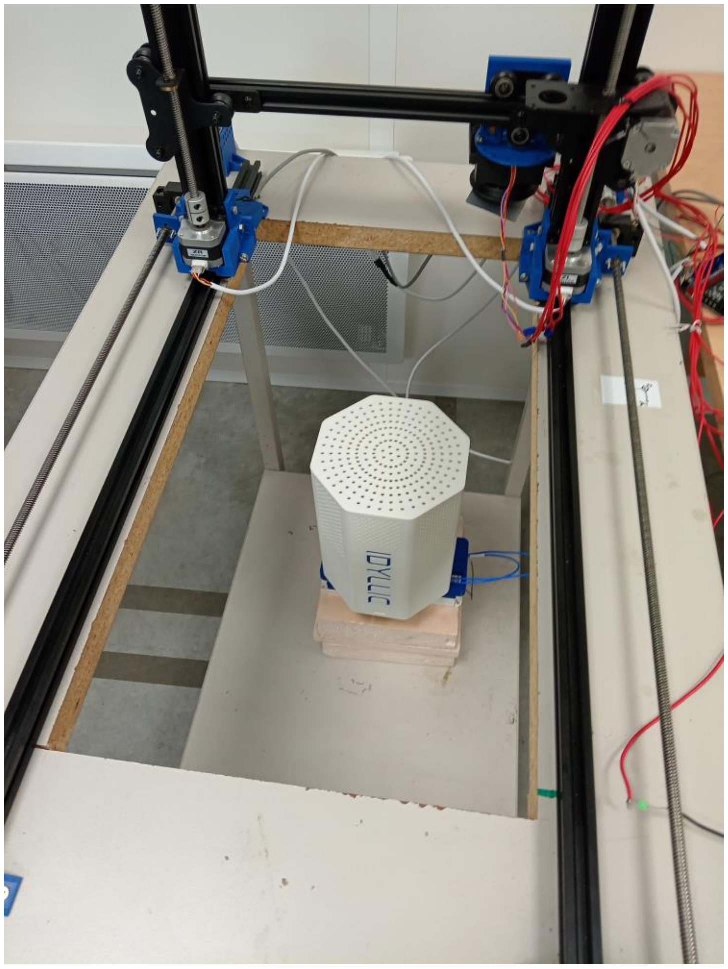
The functions of the code are pretty much the same as before, but the only difference is that i added the variables pos_delay_x , pos_delay_y, pos_delay_z in the input. Because python is continuously sending the commands we need to wait until the machine reaches the position we want before it moves at the next command.

Last Thoughts

The arduino and the extra motor can be replaced if we buy a new board with more axis available but because i had no time i tried this solution that i just showed




KORAD
VOLTAGE
CURRENT
DIGILENT
Va
Vb
Vc


IDYLLIC