California Polytechnic State University, Pomona
Department of Electrical and Computer Engineering

Intro to Microcontrollers
ECE 3301

Project Report: Drawing Robot

Group 8
Patrick Nguyen, John McDonald, Jetts Crittenden, Adam Devictoria

Professor Tamer Omar
ECE 3301
4 December 2024
Fall 2024

ABSTRACT

Our paper demonstrates the development and implementation of a T-bot plotter system that allows for movement and illustration on the x, and y-axis in order to draw images developed with GCODE. The T-bot system utilizes two stepper motors linked by a single chain. These two motors are controlled by external drivers. The experimental methodology involved in the development of our design consisted of extensive research on the type of ganty's used in other plotters/3D printers, and the code required to integrate the motors with the GCODE instructions. Key stages in development included the first step of running two motors simultaneously at different speeds, integrating an emergency stop button via interrupts, and fine-tuning the z-axis control using a stepper motor for the up and down movement of the writing utensil. We were able to get the system to execute complex movements using the GCODE and other instructions. Various challenges such as component delay, motor driver control issues, and limited access to tools and time were addressed to compact planning and design adjustments. We were able to achieve our goals demonstrating the effectiveness of a plotter design on the PIC18 platform.

TABLE OF CONTENTS

INTRODUCTION

At the beginning of this project, we discussed several ideas, in the end, we decided to create a robot that can replicate, or draw, simple images on a sheet of paper. Otherwise known as a plotter. We first determined the type of plotter to model our project on, and what kinds of components are required to purchase or 3D print. As the orders were placed time was set aside to utilize the library Maker Studio found on the second floor of the campus library. There were two main ways to interpret the images in the code but we chose to read inputs from an image file one at a time and translate them into data that can be used to control the robot via the microcontroller. We used a microcontroller and C code to control the encoding, understanding, and replication of the image. Our robot moves the drawing instrument similar to the x and y-axis motion of a 3D printer. The physical robot will inscribe the illustration guided by stepper motors and an x,y axis track utilizing belts and screw rods. The drawing instrument is controlled via a system that applies light pressure on the z-axis, perpendicular to the movement system, to apply and remove a writing utensil. Controlling the robot via C code and image understanding the machine will be able to draw shapes, lines, letters, curves and other objects that any image can be broken down into.

The scope of the project included many concepts. First, the C code is needed to accurately control the two stepper motors and the construction of the printed and purchased frame. Second was basic gcode Image file processing which converted the gcode file into C code inputs interpreted by the microcontroller and utilized to control the step motors for axis movement and drawing utensil pressure. Pairing the two together we were able to create a system that can replicate an image given to it in a gcode file.

EXPERIMENTAL METHODOLOGY

We began our experimental process by researching various designs that mirror our proposed project to give references for the parts that would be required to construct the machine as well as the code for drawing different images. We ultimately settled on a design that used stepper motors to move a gantry that can move in the x and y direction as well as a motor that moves the system in the z direction. This style of gantry is known as a T-bot which only requires a single belt for x and y operation. For programming, C was used to control the motors because we had the most familiarity with using C for the PIC microcontroller. G-code was then converted into C and was used for image processing because G-code is a programming language designed to create different shapes and objects.

Project Design:

Initially we began by deciding which of the 3 most common structures we would use. The structures we compared were the H-bot, T-bot, & Cartesian Head. The Cartesian Head uses two belts which does allow for a simpler movement control while the T-bot and H-bot use 1 belt. In the end we decided to create a T-bot design with 2 stepper motors & 1 servo connected with belts on a printed & constructed system. The T-bot gantry uses a pulley system that moves the x and y axes of the system. The stepper motors allow for full movement on the xy plane.

The two stepper motors used for the x and y operation draw 12 volts and 1.5 amps and require an external power source. To control the power drawn from the external power source we used A4988 driver to provide the correct current and voltage to the stepper motor. The driver also interfaces with the microcontroller to take inputs from the microcontroller and output the instructions to the motor. The stepper motors move in steps of 1.8 degrees, with the direction determined by the direction pin and the speed determined by the frequency. The servo motor used for z direction operation is connected directly to the microcontroller because it has a driver built in. The servo receives a PWM pulse from the PIC18F4620 and then moves at a certain angle based on the frequency of the pulse. To lift the pen, pulses with a 1 ms width are used while pulses with a 1.5 ms width lowers the pen.

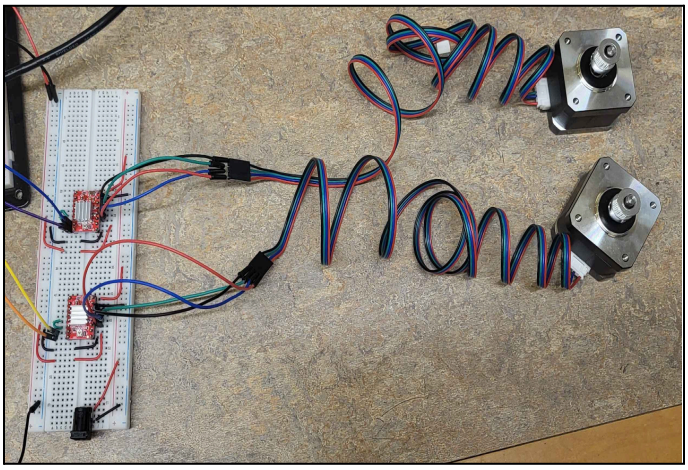
Development of Code:

STAGE 1: stage 1 of development involves being able to run 2 motors at once and at different speeds. The way A4988 drivers interface with the stepper motors is on every rising edge of the signal at the step pin, a 1.8-degree turn is output to the motor

```
void main()
{
    nRBPU = 0;
    Init_UART();
    Init_ADC();
    //parse_gcode("G0");

    TRISB = 0x00;
    TRISC = 0x00;
    M1Dir = 0;
    M2Dir = 0;
    M1Step = 0;
    M2Step = 0;
    for(int i = 0; i < 800; i++)
    {
        M2Step = (i%4==0?1:0);
        M1Step = 1;
        for (int j = 0; j < 200; j++) {}
        M1Step = 0;
        M2Step = 0;
        for (int j = 0; j < 200; j++) {}

    }
}
```



With the confirmation of being able to do this, we know we can move the gantry at different angles and draw a variety of lines and even curves

STAGE 1.5: We want to add an emergency button in case our code goes wrong so we can avoid the motors dealing damage to the system by turning when it should not be able to. We also add a button to start the program so that the program does not run unless we want it too,

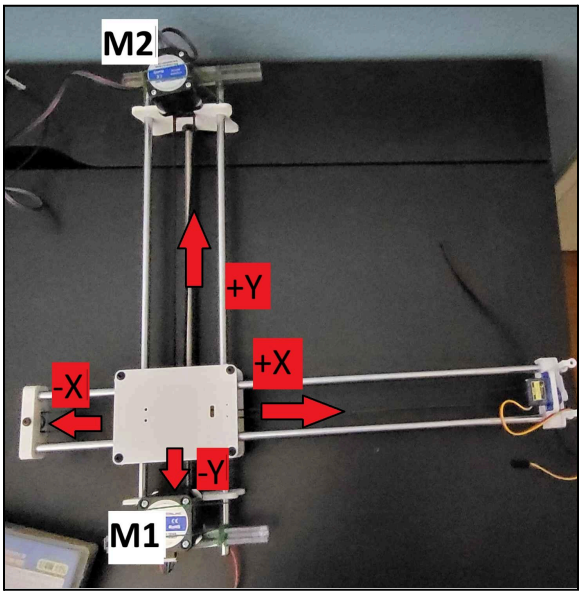
STAGE 2: We now want to test how the motors work with the belt.

There are 2 motors. They can each move 2 ways. We want to see how they move the belt by themselves. Moving only 1 motor at a time creates a movement of 45 degrees in one of the 4 quadrants resulting in the following movements.

M1 Direction	M2 Direction	Movement of the Gantry
0	-	-X-Y
1	-	+X+Y
-	0	+X-Y
-	1	-X+Y

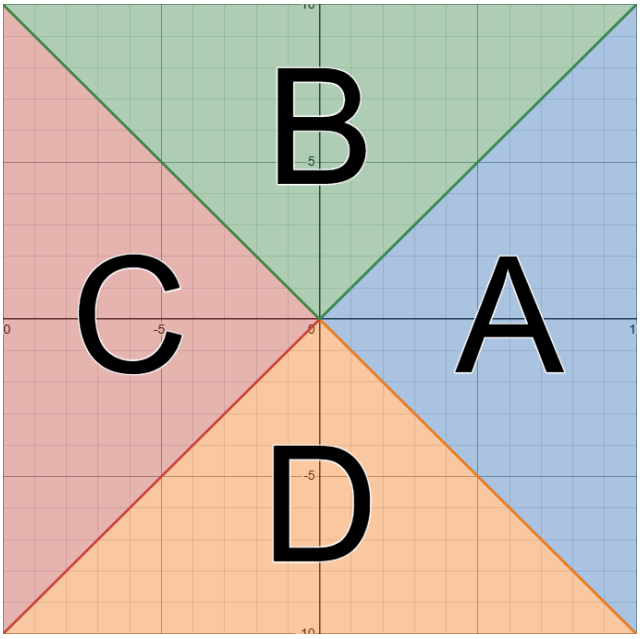
We can combine the individual movements of each motor. Running each motor at the same time adds the movements together and can create a movement on the X or Y axes. This only happens when the motors run at the same speed.

M1 Direction	M2 Direction	Movement of the gantry
0	0	-2Y
0	1	-2X
1	0	+2X
1	1	+2Y



Now we can combine the individual movements of the motors to move both at the same time but at different speeds. There are 8 cases where the speeds of the motors are different. These 8 cases allow us to move in the regions above/below the 45 degree lines and above/below the X/Y axes.

M1 Direction	M2 Direction	Movement in Region
0	0	D
0	1	C
1	0	A
1	1	B



After calculating where we want the pen head to move to we can use this table to decide what direction to apply to the A4988 direction pin for each motor and also which motor is the faster motor based on if it is above/beneath the y axis or to the left or right of x axis. We can base the fastest PWM signal on the fastest motor and send a pulse to the slower motor every N pulses of the faster motor.

STAGE 3: We can now move in XY directions. The last direction we must program is the Z-axis control. For the Z-axis, we use an SG90 motor. This motor is controlled with PWM signals. The length of the pulse determines what angle the motor will move to.

We fine-tune these numbers through trial and error using a logic analyzer.

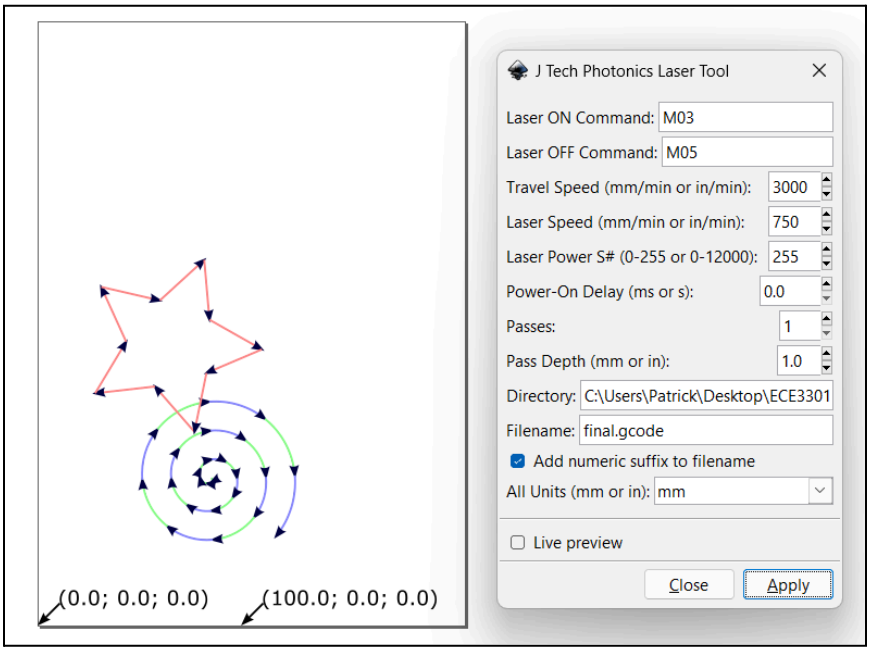
```
44  void liftServo()
45  {
46      Servo = 1;
47      for (int i = 0; i <= cycle; i++)
48      {
49          if (i == delayUp) Servo = 0;
50
51      }
52  }
53
54  void dropServo()
55  {
56      Servo = 1;
57      for (int i = 0; i <= cycle; i++)
58      {
59          if (i == delayDown) Servo = 0;
60
61      }
62  }
```

```
#define delayUp 180          //100 ms
#define delayDown 150
#define cycle 2010
```

STAGE 4: We have confirmed motor movements, but how can this make an image? We choose to use Inkscape which can turn an image into g-code.

G-code is a common code convention for the XYZ movement of machines such as laser cutters, cnc machines, and 3d printers. With its large community support it is easy to find programs that can turn images into g-code instructions. Creating our own image processing algorithms that turn images into g-code could allow for more customization to our own machine and features; however, because our machine is relatively simple, we would rather use existing programs for image to g-code conversion. We use an Inkscape extension to turn the image into g-code. We must do additional conversion of g-code into C code that runs the motors. To do this we take notes of what commands are used in the g-code.

Taking a look at a sample g-code file, we see the extension makes use of a list of g-codes commands.




```
G1 F3000
G1  X85.4226 Y72.5714
G4 P0
M03 S255
G4 P0
G1 F750.000000
G2 X85.1648 Y70.2939 I-0.9466 J-1.0462
G2 X83.0288 Y69.9256 I-1.5897 J2.8423
G2 X79.7189 Y73.0124 I0.6905 J4.0583
G2 X80.131 Y77.3591 I6.4946 J1.5772
G2 X85.7295 Y81.7152 I6.9596 J-3.1688
G2 X92.6042 Y80.5089 I1.7984 J-9.9433
G2 X98.0746 Y72.4291 I-5.6429 J-9.7123
G2 X96.006 Y62.996 I-13.4995 J-1.9829
G2 X85.4523 Y56.3869 I-12.4126 J8.09
G2 X73.4534 Y59.3423 I-2.1589 J17.0705
G2 X65.6943 Y72.3668 I10.5201 J15.0906
G2 X69.5476 Y86.9345 I20.6428 J2.3332
G2 X85.0419 Y95.8497 I17.7573 J-12.9398
G2 X102.1796 Y91.0923 I2.5074 J-24.2141
G2 X112.2544 Y73.129 I-15.353 J-20.4177
G2 X106.5893 Y53.4207 I-27.7842 J-2.6819
G2 X86.1574 Y42.1838 I-23.0743 J17.7617
G2 X63.878 Y48.7589 I-2.8569 J31.3532
G2 X51.4775 Y71.6592 I20.1673 J25.7282
```

The G0 & G1 command is for moving in a linear motion. The G2 command is for moving in an arc. The G4 command is for waiting so we can ignore this command when parsing the file. The M03 and M05 are for moving the Z-axis up and down. We can combine this with a moveToXYZ() function keeping the X and Y the same but the Z different.

STAGE 5: We have now tested the system's ability to move in a desired direction. We must now create calculations in the code to convert our desired destination to the # of pulses we will send in total. We want to create a function that takes an X and Y input and will output motor control pulses.

Now a separate module for motor movements will be created and a moveToXYZ() function will be created. This function will handle calculations and also motor movements and can be mapped to G0 & G1 commands.

To do this, the function first determines which region from STAGE 2 of the design process the point we wish to go to lays in relative to the current position. Then it calculates the distance each motor must turn to end up at that point. The distance is converted proportionally to the number of pulses we will send to the step motor.

For example, if we determine that the point is in the B region, we know that the M1 is going direction 1 and M2 is going direction 1;

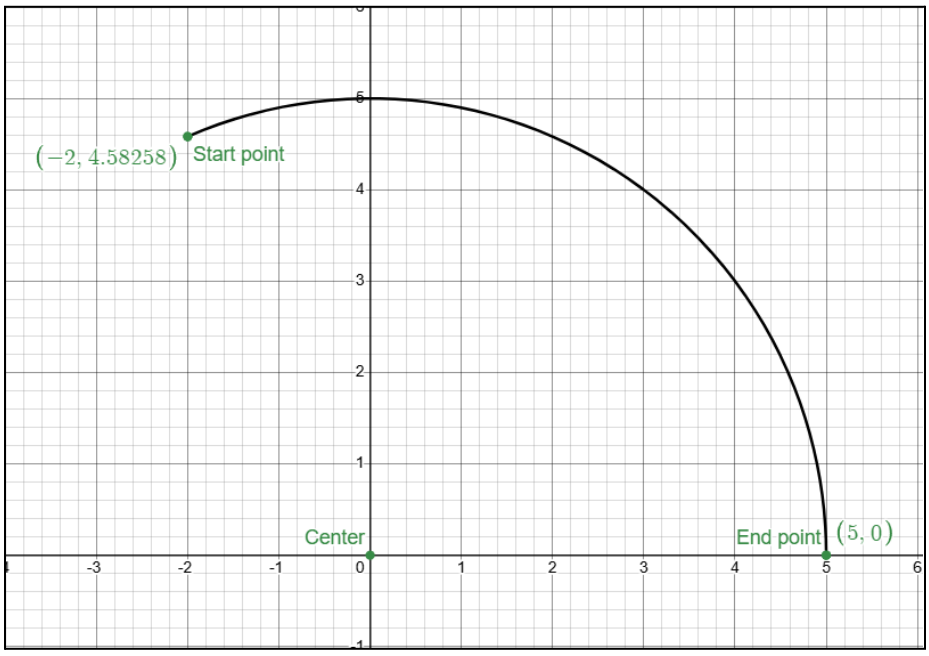
The following equation applies:

X = M1 - M2
Y = M1 + M2

Where x and y is the distance we must travel in the respective direction and M1 and M2 is how much distance a specific motor contributes to reaching x or y by spinning (a pulse to M1 moves the same amount in the x and y since it is a 45 degree line).

Solving this we can find the distances M1 must travel and the distance M2 must travel. Then we convert the distance to the number of pulses the motor will receive.

STAGE 6: Stage 4 designated a destination and moved to it in a straight line. We may also want to designate a destination but move towards it in an arc. The way that G-code does this is by designating a mid point and a destination point. The midpoint is expressed by 2 variables in the g-code format: an I and J. These are the offset from the current position to the center. I is the x offset and J is the y offset.



In this example, the resulting g-code would be:

G2 X5 Y0 I2 J-4.5858

G2 represents clockwise, X5 represents the end destination, Y0 represents end destination. The I2 represents that the x coording of the center is the starting point of -2 offset by 2 which is 0. Same for J-4.5858 but with respect to the y-coordinate.

We can use the equation of a circle to sample the points along the arc:

$$(y - y_1)^2 + (x - x_1)^2 = r^2$$

We plug in a value of x and find the y value and use the moveToXYZ() function to move to that point. To move along the arc we increment x and calculate y until we either reach one of the corners (top bottom left or right) of the circle or we reach the destination.

We create 2 functions for this: moveArcCW(x,y,z,i,j) and moveArcCCW(x,y,z,i,j). One is for clockwise movement and one for counter clockwise movement.

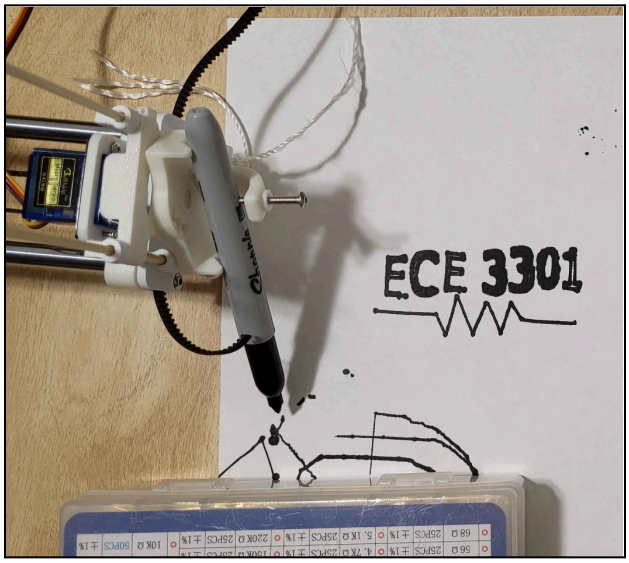
STAGE 7: Stage 7 is the final stage. This is the conversion of g-code to C. We will right another program that will do the preprocessing of g-code to C. It will take an input g-code file and produce an output C file.

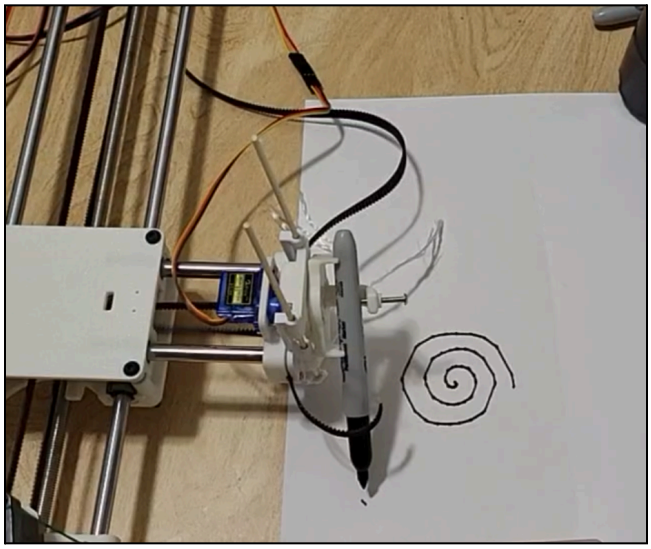
This C file will be a function composed of smaller moveToXYZ() and moveArcCW()/moveArcCCW().

When we parse the file we will check for 2 things. This will be the letter G which indicates the start of a command followed by a number which is the command number. Then we will look for the X, Y, Z, I, and J parameters that follow the G. When we read these values we will write to the C file the corresponding function that would be used.

EXPERIMENTAL RESULTS

Our design is successfully able to move in the x direction, y direction, and both at the same time using stepper motors. The design also moves in the z direction so the pen/pencil is touching the paper to draw utilizing a simple servo that moves between two preset positions, touching the paper & holding the pen up to move to the next starting line. The microcontroller is able to use the G-code provided and draw an image that is accurate to the original image that was provided. This geode is generated using a J-Tech extension in Inkscape. The geode is then placed into a file in the program which is parsed using our group made parser to convert it into C code instructions understood by the PIC18F4620 microcontroller, & the stepper motors/servo it controls via the drivers. These Code movement instructions are the conversion of the G0,G1,G2,G3 geode commands that control straight & curved motion. We combined G0 and GI into a single movement command in C code since the only difference between them is speed. This system isn't the fastest & requires a large memory in its current state but it can draw shapes, curves• letters, & numbers.





Demo Video: https://youtu.be/CqcDJ_57IfA

SUMMARY OF CHALLENGES & SOLUTIONS

Over the course of completing this project we encountered several challenges. The first challenge that gave us some trouble was the time required for items ordered online to arrive and the items being printed needing to be done during the Maker Studio's weekly hours which didn't always align nicely with the class schedules of our group members. We weren't able to completely fix this as classes and studio hours were set but we worked around it by meeting on days off to complete the construction of the robot and code testing.

Once we had the components and had begun work on learning to control the stepper motors via the microcontroller and drivers we had issues getting the motors to run smoothly in one direction, the motors would step forward and backward causing vibration rather than spinning as we were attempting to do. The solution to this issue found by testing code and researching the functionality and control of the stepper motors was that the wiring scheme was incorrect and the signal pulse was too fast. Additionally, two of the purchased drivers were broken..

A challenge we faced with construction was a lack of access to proper tools. We did not have access to manufacturing tools for the metal pieces like a saw or drill. We overcame this challenge by reworking our designs to not require that we cut the rods to a new length and utilized what we did have on hand to resize holes as necessary.

The final challenge we faced occurred during the demo, unfortunately the servo motor used to press down the drawing instrument burned so we had to demo using the pre prepared video.

CONCLUSIONS

In our project proposal we set out some deliverables as tasks we needed to complete for this project. The tasks included: an output system that can move in an xy grid, an output system that can press down to draw, an input system that reads the image, an algorithm to process the image into readable data, and an algorithm to convert the readable data into simple robotic movements.

As we have shown in our demo the robot is slow for complex images, to allow for accuracy, but still very capable of copying the images given to it via gcode. Therefore, we have accomplished what we set out to do, our robot fits within the options we left ourselves in the proposal portion of this project and accomplishes the necessary deliverables.

Overall there is room for improvement. First we could connect the microcontroller to a device with a larger storage such as a computer to allow us to draw more complex images. Second we could refine the curve functions so they draw cleaner curves compared to the lines between points on a curve we are currently using. The last improvement we would implement if we took this project further would be some kind of Mount to hold the system and the paper in addition to better mounts for the stepper motors. It is necessary but we could also use stepper motors with a souller degree per step for further accuracy.

REFERENCES

[1] “Nema 17 Bipolar 45Ncm (63.74oz.in) 1.5A 42x42x39mm 4 Wires w/ 1m Pin Connector,” www.omc-stepperonline.com. https://www.omc-stepperonline.com/nema-17-bipolar-45ncm-63-74oz-in-1-5a-42x42x3mm-4-wires-w-1m-pin-connector-17hs15-1504s-x1

[2] Last Minute Engineers, “Control Stepper Motor with A4988 Driver Module & Arduino,” Last Minute Engineers, Nov. 28, 2018. https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/

[3] A. Pieters, “SG90 Servo - Pinout -,” Feb. 15, 2022. https://www.studiopieters.nl/tower-pro-micro-servo-s9-sg90/