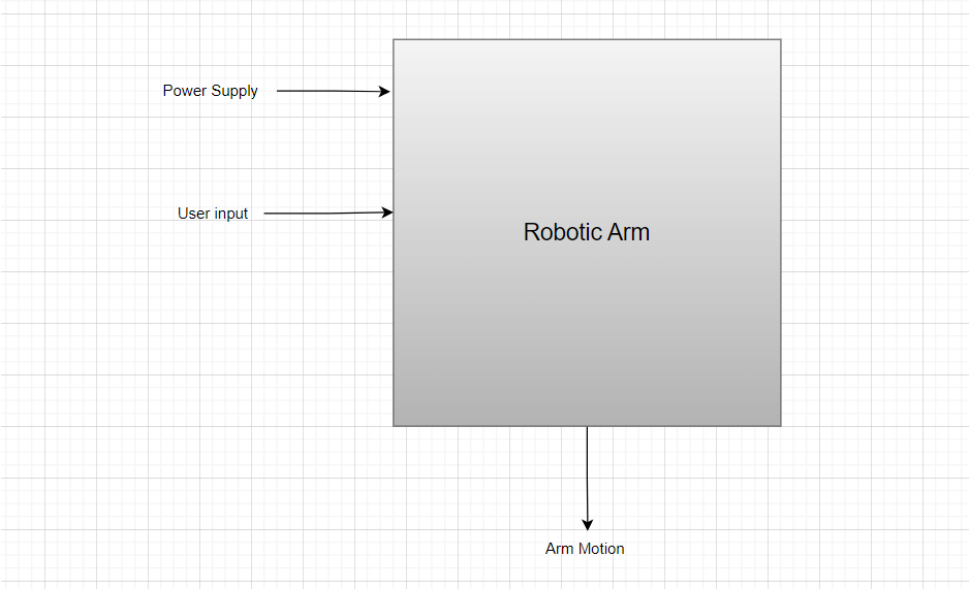
Robotic Arm 004-3

Date: 3/15/2023

Team Members: David Jepsen, Logan Nesting, Nishant Sane

System Design



The robotic arm is designed to write on 8.5xll inch piece of paper. The system is designed to be inexpensive and controlled via a commonly known programming language. The system is designed in a 2-axis SCARA design. The system is controlled via user inputs through a python user interface that utilizes G-code to guide the system's movements. The system is designed to be accurate and fast with a variable speed. Additionally, the system is designed to mount many tools, including a pen, crayon, and pencil. Additionally, the system is designed to handle multipoint inputs and then return to its starting location.

Interface Name	**Interface Type**	**Specifics**
Power supply	DC Power	- L293D chip takes 5V DC for internal logic translation - Arm motors (NEMA 17 stepper motors) take 12V DC
User Input	G-Code	- The user will send in G-codes that will correspond with specific arm motions and functionality, according to the engineering requirements - Arm motion will have the option to specify the speed the tool will move

Arm motion	Movement	X motion Y motion

Testing plan

Customer Requirement: The system should be fast.

Objective: The serial should display a speed greater than 4 in/sec.

1) Input a speed into the slider window
2) Check that the robot will output this speed in the Python terminal.

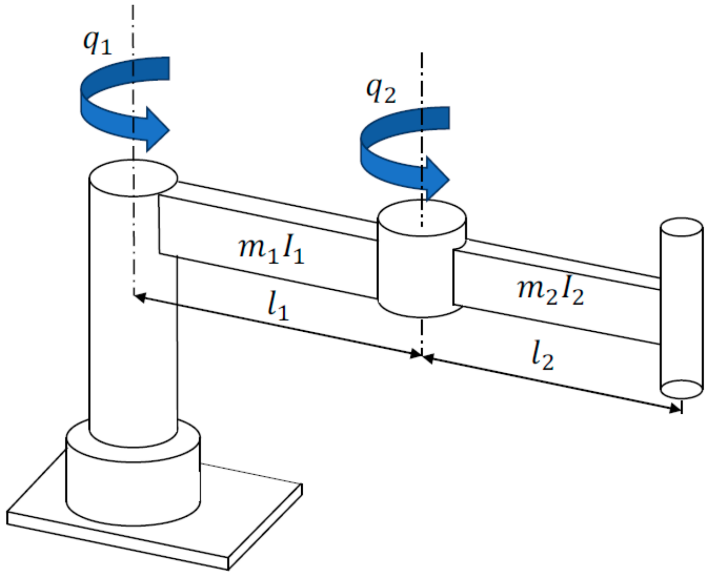
Customer Requirement: The system must be accurate.

Objective: The system should be able to draw an approximate 10 in line with +- 0.25 inch variation maximum.

1) Input coordinates (7,7) which is 9.89 inches.
2) The robot will output degrees
3) Manually show through forward kinematics that this is actually (7,7).

Customer Requirement: The system needs to be inexpensive and manageable to manufacture.

Objective: Prove the robot has a SCARA topology.



1. Observe the robot and the number of joints
2. Check if the joints articulate
3. Check if the robot is similar to the reference photo.

Customer Requirement: The system must have a commonly known interface.

Objective: Show that the Python PyQT GUI translates user inputs to Gcode commands and sends them to Arduino via serial.

1. Plug in Arduino to the computer via USB and wire the motors to PCB, PCB to Arduino
2. Verify/upload the Arduino script, then run the Python program
3. Follow the prompts given by the Python program to enter coordinate input, change the units, drawing mode, utensil, or end the program.

Customer Requirement: The system must use different types of writing tools.

Objective: Show that the robot can hold a pen, pencil, or crayon and is mounted in under 15 seconds.

1. Select the tool change option in GUI
2. Open up the tool holder by hand
3. Remove the tool and insert a new tool
4. Check the amount of time elapsed

Customer Requirement: The system must be able to handle multi-point inputs.

Objective: Input up to 3 points in the GUI without breaking out of program execution.

1. Open the Python GUI
2. Select to add points
3. Select the number of points to add (1, 2, or 3)
4. Tune the coordinate sliders to the desired locations and close the slider window to send coordinates to serial
5. Repeat step 4 until the desired number of coordinates is entered
6. Enter other commands if desired

Customer Requirement: The system must have a variable speed.

Objective: vary the speed of the program.

1. Open the python GUI
2. Tune the speed slider present on the GUI
3. Then close the slider window, and the signal will be sent
4. The speed will be displayed in the python terminal to be verified

System Requirements

1. **Customer Requirement: The system should be fast.**
   Engineering Requirement: The system must draw faster than 4 inches per second.
   - Verification:https://youtu.be/fM_iYZbGJhc

2. **Customer Requirement: The system must be accurate.**
   Engineering Requirement: The system must draw a 10 inch straight line +/- .25 inch. This includes both the overall length of the line and ensuring the line does not vary more than .25 inches of being perfectly straight.
   - Verification: https://youtu.be/F0tB6ZRYiP8

3. **Customer Requirement: The system needs to be inexpensive and manageable to manufacture.**
   Engineering Requirement: The robotic arm will use a SCARA topology, with two rotating joints to control arm actuation.
   - Verification:https://www.youtube.com/shorts/lHMDzx-CvMY

4. **Customer Requirement: The system must have a commonly known interface.**
   Engineering Requirement: Controlling commands will be input as G-code commands. These commands must be made available within the Python or MATLAB GUI. G0, G1, G90, G91, G20, G21, M2, M6, M72.
   - Verification: https://youtu.be/V1HdTCXGmgk

5. **Customer Requirement: The system must use different types of writing tools.**
   Engineering Requirement: Upon receiving an M6 command, the machine operator must mount a crayon, pen, or pencil within 15 seconds.
   - Verification:https://youtube.com/shorts/25zxZr2BN1A?feature=share

Additional Engineering Requirements

6. **Customer Requirement: The system must be able to handle multi-point inputs.**
   Engineering Requirement: The system on a specific command within the GUI will display the option to move to 3 user-selected points. The system will respond to

user-selected points to show the full range of motion. Upon reaching the final location, the system will return to its original starting point.

- Verification:https://youtu.be/sQrXkDogHbM

7. **Customer Requirement: The system must have a variable speed.**
   Engineering Requirement: The system will allow users to choose a drawing speed between 1 inch per second and 5 inches per second. The system will respond to this input, drawing at the given speed. This drawing speed can be tested by timing how long the system takes to draw the 10-inch line.
   - Verification:https://youtu.be/sQrXkDogHbM
     (same video as Engineering requirement 6)

System Artifacts

Math

Length of arms

$l_1 = 9.78$ (inches)

$l_2 = 9.78$ (inches)

Absolute mode cos calculation:

$$cos(\theta_2) = \frac{(x_{target} - x_{current})^2 + (y_{target} - y_{current})^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

Relative mode cos calculation:

$$cos(\theta_2) = \frac{x_{target}^2 + y_{target}^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

We use the respective cos calculation for whichever mode we are in.

$$sin(\theta_2) = \sqrt{1 - cos^2(\theta_2)}$$

$$\theta_2 = arctan2(sin(\theta_2), cos(\theta_2))$$

$$k_1 = l_1 + l_2 cos(\theta_2)$$

$$k_2 = l_2 sin(\theta_2)$$

$$\gamma = arctan2(k_2, k_1)$$

$$\theta_1 = arctan2(y_{target}, x_{target}) - \gamma$$

Code

The arm operates on user-specified input for various G Code parameters which are obtained via a Python script. When the Python code is running, the user is asked to specify whether the arm should operate in “Absolute” (G90) or “Relative” (G91) mode. The script also plots between 1 and 3 coordinates (G0, G1), and whether the distances should be measured in inches (G20) or millimeters (G21). A slider (via the PyQT library) obtains the desired coordinates and drawing speed. The user is prompted to select if they will change the drawing tool (M6) and if they are to end the program (M2). After going through G20, and G21 commands, the coordinates are run through inverse kinematic calculations, and the resulting angle calculations are then sent to Arduino (via serial). The byte string sent to Arduino is formatted: ‘angle1 angle2 draw_speed x y ‘. When the user chooses to operate in G90 or G91, a byte string of ‘absolute’ or ‘relative’ is written to Arduino.

Once serial communications are opened in the Arduino IDE, the Uno reads in any incoming byte strings and converts them to a C-string. Once in a C-string, the IDE saves each character into temporary arrays if they are numerical, ‘a’ to ‘z’, a negative sign, or a decimal point. For input strings into the Arduino that have multiple parameters, the parameters are saved to different variables by looping through the string and counting the number of spaces. The number of spaces counted in the loop determines which parameter is which. Once the parameters are assigned to their respective variables, they are plugged into the AccelStepper library to drive the arm’s motors.

Arduino Code:

```
#include <Arduino.h>
#include <AccelStepper.h>
#include <MultiStepper.h>
#include "SerialTransfer.h"
#include <math.h>


const float Pi = 3.14159;
float x_target = 0; // target x-position
float y_target = 0; // target y-position
float x_current = 0;
float y_current = 0;
const long l1 = 9.78; // length of first section of arm
const long l2 = 9.78; // length of second section of arm
long positions[2];
long lin_pos[2];
bool relativeMode = false;
```

```
float speed = 4;
float speed1 = 0;
float speed2 = 0;
bool close = false;
bool commandGiven = false;
float sec = 0;
bool inchMode = true;
bool absoluteMode = true;

// Number of steps per output rotation
const int stepsPerRevolution = 200;

// Create Instance of Stepper library for each motor
AccelStepper M1(AccelStepper::FULL4WIRE, 12, 11, 10, 9);
AccelStepper M2(AccelStepper::FULL4WIRE, 8, 7, 6, 5);

MultiStepper ARM;

void setup()
{
  // initialize the serial port:
  Serial.begin(9600);

  // initialize the max speed at 60 rpm:
  M1.setMaxSpeed(60);
  M2.setMaxSpeed(60);

  M1.setCurrentPosition(0);
  M2.setCurrentPosition(0);

  M1.setAcceleration(40);
  M2.setAcceleration(40);

  ARM.addStepper(M1);
  ARM.addStepper(M2);
}

double angle2(double x, double y){

  double a2_rad = acos((x*x + y*y - l1*l1 - l2*l2)/(2*l1*l2));
  double a2_deg = a2_rad*180/Pi;
  return a2_deg;
}
```

```
double angle1(double x, double y, double a2){

  float k1 = l1 + l2*cos(angle2(x, y));
  float k2 = l2*sin(angle2(x, y));
  float gamma = atan2(k2, k1);
  double a1_rad = atan2(y, x) - gamma;
  double a1_deg = a1_rad*180/Pi;
  return a1_deg;
}

void drive(float x_c, float x_t, float y_c, float y_t, bool mode, float spd){
  if(relativeMode){
    Serial.println("Drives in relative");
    positions[0] = angle1(x_t-x_c, y_t-y_c, angle2(x_t-x_c, y_t-y_c))*stepsPerRevolution/360;
    positions[1] = angle2(x_t-x_c, y_t-y_c)*stepsPerRevolution/360;
    sec = sqrt(sq(x_target - x_current)+sq(y_target - y_current))/speed;
  }
  else if(!relativeMode){
    Serial.println("Drives in absolute");
    positions[0] = angle1(x_t, y_t, angle2(x_t, y_t))*stepsPerRevolution/360;
    positions[1] = angle2(x_t, y_t)*stepsPerRevolution/360;

    sec = sqrt(sq(x_target)+sq(y_target))/speed;
  }
  ARM.moveTo(positions);

  Serial.print("Positions[0] (steps): ");
  Serial.println(positions[0]);

  Serial.print("Positions[1] (steps): ");
  Serial.println(positions[1]);


  M1.setSpeed(spd);
  M2.setSpeed(spd);
  Serial.print("Speed1: ");
  Serial.println(speed1);
  Serial.print("Speed2: ");
  Serial.println(speed2);
  Serial.print("Expected time spent moving: ");
  Serial.println(sec);
  long t1 = millis();

  ARM.runSpeedToPosition();
```

```
  long t2 = millis();
  Serial.print("Actual time spent moving: ");
  Serial.println(t2 - t1);
}

void loop()
{
  String ser_data;
  int idx = 0;
  char temp_xy[2];
  char byteIn;
  String in_str;
  String byte_data[45];

  while(Serial.available()){
    in_str = Serial.readString(); // read in byte from buffer (once read, byte removed from buffer)
    in_str.trim();
    ser_data = in_str;
    idx++;
    commandGiven = true;
  }

  if(commandGiven){
    Serial.print("ser_data is: ");
    Serial.println(ser_data);
    char cmd[ser_data.length()];
    int cmd_idx = 0;
    for(int i = 0; i < ser_data.length() - 1; i++){ // read command and store it to a c-string
      cmd[cmd_idx] = ' ';
      if(ser_data[i] == 'G' || ser_data[i] == 'M' || ser_data[i] == 'X' || ser_data[i] == 'Y' || ser_data[i]
== 'S' || ser_data[i] == ' ' || (ser_data[i] >= '0' && ser_data[i] <= '9')){
        cmd[cmd_idx] = ser_data[i];
        cmd_idx++;
      }
    }
    Serial.print("G/M command: ");
    Serial.println(cmd);
    Serial.println(cmd[0]);
    Serial.println(cmd[1]);
    Serial.println(cmd[2]);
    if(cmd[0] == 'G' && cmd[1] == '0' && cmd[2] == '1'){ // G01 command
      for(int j = 0; j < cmd_idx; j++){
        if(cmd[j] == 'X'){
```

```
      temp_xy[0] = cmd[j+1];
      temp_xy[1] = cmd[j+2];
      x_target = atoi(temp_xy);
      Serial.print("x_target: ");
      Serial.println(x_target);
    }
    if(cmd[j] == 'Y'){
      temp_xy[0] = cmd[j+1];
      temp_xy[1] = cmd[j+2];
      y_target = atoi(temp_xy);
      Serial.print("y_target: ");
      Serial.println(y_target);
    }
    if(cmd[j] == 'S'){
      temp_xy[0] = cmd[j+1];
      temp_xy[1] = ' ';
      speed = atoi(temp_xy);
      Serial.print("s_target: ");
      Serial.println(speed);
    }
  }
  drive(x_current, x_target, y_current, y_target, relativeMode, speed);
}
if(cmd[0] == 'G' && cmd[1] == '2'){
  if(cmd[2] == '0' && !inchMode){ // G20 command (inches)
    Serial.println("Receives G20 Command");
    x_current = x_current*25.4;
    Serial.print("x_current: ");
    Serial.println(x_current);
    x_target = x_target*25.4;
    Serial.print("x_target: ");
    Serial.println(x_target);

    y_current = y_current*25.4;
    Serial.print("y_current: ");
    Serial.println(y_current);
    y_target = y_target*25.4;
    Serial.print("y_target: ");
    Serial.println(y_target);

    speed = speed*25.4;
    inchMode = true;
  }
  if(cmd[2] == '1' && inchMode){ // G21 command (mm)
```

```
    Serial.println("Receives G21 Command");
    x_current = x_current/25.4;
    Serial.print("x_current: ");
    Serial.println(x_current);
    x_target = x_target/25.4;
    Serial.print("x_target: ");
    Serial.println(x_target);

    y_current = y_current/25.4;
    Serial.print("y_current: ");
    Serial.println(y_current);
    y_target = y_target/25.4;
    Serial.print("y_target: ");
    Serial.println(y_target);

    speed = speed/25.4;
    inchMode = false;
  }
}
if(cmd[0] == 'G' && cmd[1] == '9'){
  if(cmd[2] == '0' && !absoluteMode){ // G90 command (Absolute mode)
    Serial.println("Receives G90 Command");
    relativeMode = false;
    x_current = x_target;
    Serial.print("x_current: ");
    Serial.println(x_current);

    y_current = y_target;
    Serial.print("y_current: ");
    Serial.println(y_current);

    absoluteMode = true;
  }
  if(cmd[2] == '1' && absoluteMode){ // G91 command (Relative mode)
    Serial.println("Receives G91 Command");
    relativeMode = true;
    x_current = x_current + x_target;
    Serial.print("x_current: ");
    Serial.println(x_current);

    y_current = y_current + y_target;
    Serial.print("y_current: ");
    Serial.println(y_current);
```

```
      absoluteMode = false;
    }
   }
   if(cmd[0] == 'M' && cmd[1] == '0'){
    if(cmd[2] == '2'){ // M02 command (Exit Program)
     Serial.println("End Program");
     delay(500);
     exit(0);
    }
    else if(cmd[2] == '6'){ // M06 command (Change Tool)
     Serial.println("Changing Tool");
     delay(15000);
    }
   }
  }

  commandGiven = false;
  delay(500);
  idx = 0; // Reset serial.read index for next buffer input
}
```

Python Code:

```
import struct

from PyQt6.QtWidgets import (QApplication, QLabel,
                             QVBoxLayout, QWidget, QSlider, QDockWidget)
from PyQt6.QtCore import Qt
import sys
import time
import serial
import math


def getSpeed():  # Accessor for the speed slider value
   return window.speed_slider.value()


def getX():  # Accessor for the X coordinate
   return window.x_slider.value()


def getY():  # Accessor for the Y coordinate
   return window.y_slider.value()
```

```
def set_default():  # Default measure and operation settings
   default = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600, timeout=1,
writeTimeout=1)  # Open serial connection to port at 9600 baud rate
   print("Default Settings: Inches (G20) and Absolute mode (G90) \n")
   default.write(bytes('G90', encoding='utf-8'))
   time.sleep(2)
   default.close()


def g01():  # Draws straight lines
   num_coord = input("How many coordinates would you like to plot (Enter '1',
'2' or '3')?: ").strip()
   while 1:
       if num_coord == '1' or num_coord == '2' or num_coord == '3':
           for _ in range(int(num_coord)):
               app.exec()  # runs the app that displays the slider window
               window.show()  # displays the slider window
               x_coord = getX()  # Gets X Coordinate
               y_coord = getY()  # Gets Y Coordinate
               speed = getSpeed()  # Gets Speed
               param_list = [x_coord, y_coord, speed, 0]  # Place user input
parameters in list to send to Arduino
               len1 = 9.78  # Joint 1 length = 9.78 inches
               len2 = 9.78  # Joint 2 length = 9.78 inches
               angle2 = math.acos((x_coord * x_coord + y_coord * y_coord - len1
* len1 - len2 * len2) / (2 * len1 * len2))  # IK calcs for Joint 2 angle
               angle1 = math.atan(y_coord / x_coord) - math.atan((len2 *
math.sin(angle2)) / (len1 + len2 * math.cos(angle2)))  # IK calcs for Joint 1
angle
               angle1 = math.degrees(angle1)  # Change angles from radians to
degrees and show angles
               angle2 = math.degrees(angle2)
               print("Angle 1: ", angle1)
               print("Angle 2: ", angle2)
               print("Coordinate #", _ + 1, "Input Parameters Chosen: ")
               print("Speed: ", param_list[2])  # Displays the user-chosen
speed in the terminal
               print("X: ", param_list[0])  # Displays the user-chosen X
coordinate in the terminal
               print("Y: ", param_list[1])  # Displays the user-chosen Y
coordinate in the terminal
               G01_str = "G01 " + "X" + str(x_coord) + " Y" + str(y_coord) + "
S" + str(speed)
               link = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600,
timeout=1, writeTimeout=1)  # Open serial connection to port at 9600 baud rate
               time.sleep(2)
               link.write(bytes(G01_str, encoding='utf-8'))
```

```
                print("Buffer contents: ", G01_str.encode(encoding='ascii',
errors='strict'))
                print("Wait for the motors to reach your desired location \n")
                time.sleep(math.sqrt(x_coord*x_coord + y_coord*y_coord)/speed)
                link.close()  # closes the usbserial port
                time.sleep(0.5)

            break


def g20_21(units):  # Changes units from in to mm or vice versa
    link = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600, timeout=1,
writeTimeout=1)  # Open serial connection to port at 9600 baud rate
    if units == 'in':  # Change units to inches
        link.write(bytes('G20', encoding='utf-8'))
        print("Buffer contents: ", 'G20'.encode(encoding='ascii',
errors='strict'))
    elif units == 'mm':  # Change units to mm
        link.write(bytes('G21', encoding='utf-8'))
        print("Buffer contents: ", 'G21'.encode(encoding='ascii',
errors='strict'))
    time.sleep(1)
    link.close()  # closes the usbserial port


def g90_91(rel_abs_mode):  # Operates in Absolute or Relative mode
    link = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600, timeout=1,
writeTimeout=1)  # Open serial connection to port at 9600 baud rate
    if rel_abs_mode == 'relative':  # Change to rel
        link.write(bytes('G91', encoding='utf-8'))
        print("Buffer contents: ", 'G91'.encode(encoding='ascii',
errors='strict'))
    elif rel_abs_mode == 'absolute':  # Change to abs
        link.write(bytes('G90', encoding='utf-8'))
        print("Buffer contents: ", 'G90'.encode(encoding='ascii',
errors='strict'))
    time.sleep(1)
    link.close()  # closes the usbserial port


def m2():  # Sends command to end program
    link = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600, timeout=1,
writeTimeout=1)  # Open serial connection to port at 9600 baud rate
    time.sleep(2)
    link.write(bytes('M02', encoding='utf-8'))
    print("Buffer contents: ", 'M02'.encode(encoding='ascii', errors='strict'))
    time.sleep(1)
    link.close()
    print("Program Ended")
```

```
def m6():  # Sends command to change tool
   link = serial.Serial('/dev/cu.usbserial-14410', baudrate=9600, timeout=1,
writeTimeout=1)  # Open serial connection to port at 9600 baud rate
   link.write(bytes('M06', encoding='utf-8'))
   print("Buffer contents: ", 'M06'.encode(encoding='ascii', errors='strict'))
   print("Changing drawing tool - wait 15 seconds...")
   time.sleep(15)
   link.close()


class Window(QWidget):  # Defines class for a window of widgets
   def __init__(self):
       super().__init__()

       # Resize, and label the main window
       self.resize(1000, 1000)
       self.setWindowTitle("User Input")
       self.setGeometry(300, 300, 800, 600)
       self.label = QLabel(self)
       self.label.move(340, 50)
       self.label.alignment()

       # Create a dock so the 3 input sliders can be on the same widget
       dock = QDockWidget(self)
       dock.setWindowTitle("Drawing Speed and Coordinates")
       dock.setGeometry(100, 100, 600, 300)
       widget = QWidget(self)
       layout = QVBoxLayout(self)

       # Speed slider window initialization
       self.speed_slider = QSlider(Qt.Orientation.Horizontal, self)
       self.speed_slider.setGeometry(300, 300, 200, 200)
       self.speed_slider.setMinimum(0)
       self.speed_slider.setMaximum(5)
       self.speed_slider.setValue(1)  # Default value after initialization
       self.speed_slider.setTickPosition(QSlider.TickPosition.TicksBelow)
       self.speed_slider.setTickInterval(1)
       self.speed_slider.valueChanged.connect(self.changedValue)  # Updates
variable when the slider is on it
       self.speed_slider.valueChanged.connect(self.speedLabel)

       # X coordinate slider window initialization
       self.x_slider = QSlider(Qt.Orientation.Horizontal, self)
       self.x_slider.setGeometry(300, 300, 200, 200)
       self.x_slider.setMinimum(4)
       self.x_slider.setMaximum(12)
       self.x_slider.setValue(4)  # Default value after initialization
```

```
        self.x_slider.setTickPosition(QSlider.TickPosition.TicksBelow)
        self.x_slider.setTickInterval(1)
        self.x_slider.valueChanged.connect(self.changedValue)  # Updates
variable when the slider is on it
        self.x_slider.valueChanged.connect(self.xLabel)

        # Y coordinate slider window initialization
        self.y_slider = QSlider(Qt.Orientation.Horizontal, self)
        self.y_slider.setGeometry(300, 300, 200, 200)
        self.y_slider.setMinimum(4)
        self.y_slider.setMaximum(15)
        self.y_slider.setValue(4)  # Default value after initialization
        self.y_slider.setTickPosition(QSlider.TickPosition.TicksBelow)
        self.y_slider.setTickInterval(1)
        self.y_slider.valueChanged.connect(self.changedValue)  # Updates
variable when the slider is on it
        self.y_slider.valueChanged.connect(self.yLabel)

        # Creates/assigns/places labels for X, Y, and Speed sliders
        self.xlabel = QLabel("X coordinate", self)
        self.xlabel.move(100, 275)
        self.ylabel = QLabel("Y coordinate", self)
        self.ylabel.move(100, 350)
        self.slabel = QLabel("Drawing Speed", self)
        self.slabel.move(100, 200)
        self.xlabel.setBuddy(self.x_slider)
        self.ylabel.setBuddy(self.y_slider)
        self.slabel.setBuddy(self.speed_slider)

        # Adds the sliders to the widget layout and adds the layout to the dock
        layout.addWidget(self.speed_slider)
        layout.addWidget(self.x_slider)
        layout.addWidget(self.y_slider)
        widget.setLayout(layout)
        dock.setWidget(widget)

    def changedValue(self):  # Updates the variable based on slider position
        size = self.speed_slider.value()
        self.label.setText(str(size))

    def xLabel(self):  # Label so the user knows what X value they are selecting
        self.label.setText("X coordinate: " + str(self.sender().value()))
        self.label.adjustSize()  # Expands label size as numbers get larger

    def yLabel(self):  # Label so the user knows what Y value they are selecting
        self.label.setText("Y Coordinate: " + str(self.sender().value()))
        self.label.adjustSize()  # Expands label size as numbers get larger
```

```
    def speedLabel(self):  # Label so the user knows what Speed value they are
selecting
        # print(self.sender().value())
        self.label.setText("Speed (inches/second): " +
str(self.sender().value()))
        self.label.adjustSize()  # Expands label size as numbers get larger


app = QApplication(sys.argv)

window = Window()  # Default constructor for the window for initialization
window.show()  # Makes the window appear on the screen

set_default()  # Set default measurements and drawing mode

while 1:
    plot = input("Add more points (Enter 'yes' or 'no')?: ").strip()
    if plot == 'yes':  # If user wants to choose speed/coordinates
        g01()
    tool_change = input("Change the drawing tool (Enter 'yes' or 'no')?:
").strip()  # M6 Command
    if tool_change == 'yes':  # Drawing tool changed
        m6()
    change_units = input("Should the units be changed (Enter 'yes' or 'no')?
Measured in inches by default: ")
    if change_units == 'yes':
        in_mm = input("Should the units be measured in inches or millimeters
(Enter 'in' or 'mm')?: ")
        if in_mm == 'in':
            g20_21('in')
        elif in_mm == 'mm':
            g20_21('mm')
    mode = input("Should we change the plotting mode (Enter 'yes' or 'no')?
Plotting mode is absolute by default: ")
    if mode == 'yes':
        rel_abs = input("Change to 'relative' or 'absolute'?: ")
        if rel_abs == 'absolute':
            g90_91('absolute')
        elif rel_abs == 'relative':
            g90_91('relative')
    done = input("End the Program (Enter 'yes' or 'no')?: ")  # M2 Command (End
Program)
    if done == 'yes':
        m2()
        break

sys.exit()  # ends program
```

Obstacles and Testcode

The main obstacle we have encountered in this project is that the robot arm is not moving properly when trying to drive it using the MultiStepper library. First, we would like to demonstrate some initial tests:
We have eliminated the possibility of a PCB/wiring issue since we have tested regular Stepper code on one motor at a time:

```
// Include the Arduino Stepper Library
#include <Stepper.h>

// Number of steps per output rotation
const int stepsPerRevolution = 200;

// Create Instance of Stepper library
Stepper myStepper(stepsPerRevolution, 12, 11, 10, 9);


void setup()
{
	// set the speed at 20 rpm:
	myStepper.setSpeed(20);
	// initialize the serial port:
	Serial.begin(9600);
}

void loop()
{
	// step one revolution in one direction:
	Serial.println("clockwise");
	myStepper.step(stepsPerRevolution);
	delay(500);

	// step one revolution in the other direction:
	Serial.println("counterclockwise");
	myStepper.step(-stepsPerRevolution);
	delay(500);
}
```

When tested on one motor at a time, it works perfectly. Another follow-up test was done to drive 2 motors at 1 step alternating in a loop:

```
// Include the Arduino Stepper Library
#include <Stepper.h>

// Number of steps per output rotation
const int stepsPerRevolution = 200;

// Create Instance of Stepper library
```

```
Stepper myStepper(stepsPerRevolution, 12, 11, 10, 9);
Stepper secStep(stepsPerRevolution, 8, 7, 6 ,5)


void setup()
{
  // set the speed at 20 rpm:
  myStepper.setSpeed(20);
  // initialize the serial port:
  Serial.begin(9600);
}

void loop()
{
  // step one revolution in one direction:
  for  (int i=0; i<200; i++) {
    myStepper.step(1);
    secStep.step(-1);
  }
}
```

However, this test did not work, and the motors made some noises but did not make the proper movement. We estimate that this is because the motors need time to ramp up the torque necessary to accurately move, but since we issue such a small step command, the motor does not have time to do this and ends up failing to perform the movement sequence. This essentially eliminates the possibility of driving the motors pseudo-simultaneously using the regular Stepper library and implementing a double loop with proper step and speed calculations.
We also wrote a much simpler Multistepper test code in order to test the runSpeedToPosition function.

```
#include <AccelStepper.h>
#include <MultiStepper.h>

// define steppers (Note: NEMA 17 12V bipolar stepper motors have 200
steps per revolution)
AccelStepper M1(AccelStepper::FULL4WIRE, 12, 11, 10, 9);
AccelStepper M2(AccelStepper::FULL4WIRE, 8, 7, 6, 5);

// define multistepper object
MultiStepper ARM;

unsigned long t1;
```

```
unsigned long t2;
long positions[2];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  // set max speed of motors at 200 steps per second
  M1.setMaxSpeed(200);
  M2.setMaxSpeed(200);

  // set current positions of motors to 0 initially
  M1.setCurrentPosition(0);
  M2.setCurrentPosition(0);

  // set both motors to run at 50 steps per second
  M1.setSpeed(50);
  M2.setSpeed(50);

  // add steppers to ARM
  ARM.addStepper(M1);
  ARM.addStepper(M2);
}

void loop() {
  // put your main code here, to run repeatedly:

  // M1 should run 180 degrees clockwise, M2 should run 180 degrees
counterclockwise
  positions[0] = 100;
  positions[1] = -100;
  ARM.moveTo(positions);

  // execute motor operation, output time operation takes
  t1 = millis();
  ARM.runSpeedToPosition();
  t2 = millis();
  Serial.println(t2-t1);

  exit(0);
```

```
}
```

In both this testcode and the actual code for this robot, the robot arm fails to drive. We estimate this to be an issue either with the limitations of our L293D IC and/or the MultiStepper function runSpeedToPosition(). What we observe through the Serial monitor is an irregular time of operation taken by the runSpeedToPosition function, combined with the robot arm moving a tiny bit before stopping and generating a strange noise from the motors, along with a nearly 1A of current being drawn from the DC power supply. We suspect that since the L293D ICs have a maximum output current of ±600mA, they are forcibly shutting off due to an unexpectedly high current being drawn. We also suspect that the runSpeedToPosition function is not working properly, since the time the function takes to run is always much shorter than the calculated time of operation. In the end, we resorted to the trial and error approach to completely eliminate the possibility of a PCB/wiring issue by testing all 24 permutations of the Arduino pin ordering within the AccelStepper motor definitions, and all of them were a failure in the test code, returning a 50 ms value inside the Serial monitor, while the actual operation should take 2 seconds. A similar effect is observed within the actual code used.

Arm design

The arm is designed in two parts, the first and second joints. The first arm is 290 mm in length, with the joints spaced out 248.412mm, as seen in figure 2 and figure 3. The first joint comprises the first arm unit, 8 mm bar, bar mount, 20 teeth gear, 40 teeth gear, a timing belt, ball bearing, stepper motor, and stepper motor mount. The first arm is connected directly to the enclosure, from which the first stepper motor drives it via a belt and gear system marked in blue in figure 1. From the enclosure, the 8 mm bar is attached and held by a ball bearing mounted to the enclosure. On this 8mm bar is a 40 teeth gear driven by the first Nema 17 motor attached via a timing belt. The 8 mm bar then extends up and is mounted to the arm via an 8mm rail guide support attached to a stepper motor L bracket. The Bracket is attached to the bottom of the component via screws. Due to space, the Nema 17 motor will be mounted above the arm via bolts and nuts to allow ample room for mounting and adjustment, as seen in red in figure 1. On the Nema 17 motor shaft, a 20-teeth gear is attached to drive the second arm via a timing belt.

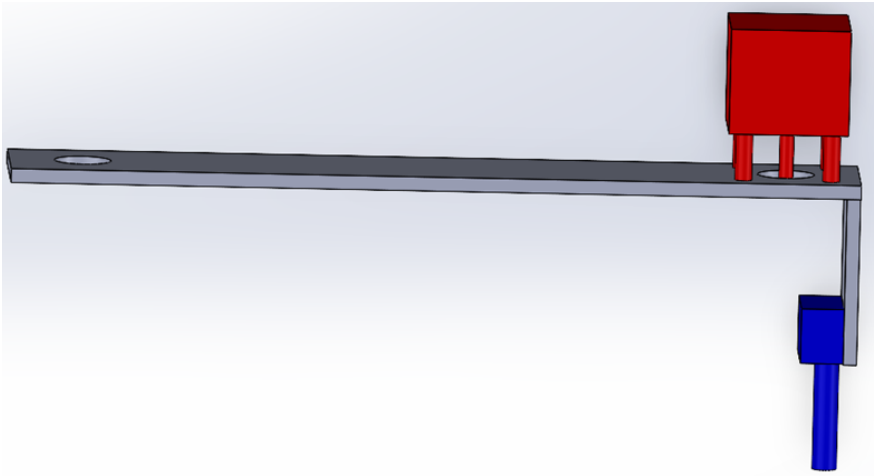


Figure 1: first arm set up

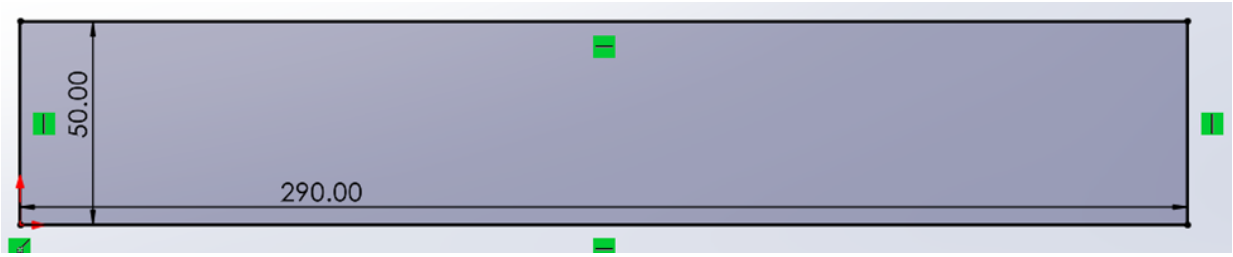


Figure 2: first arm measurements

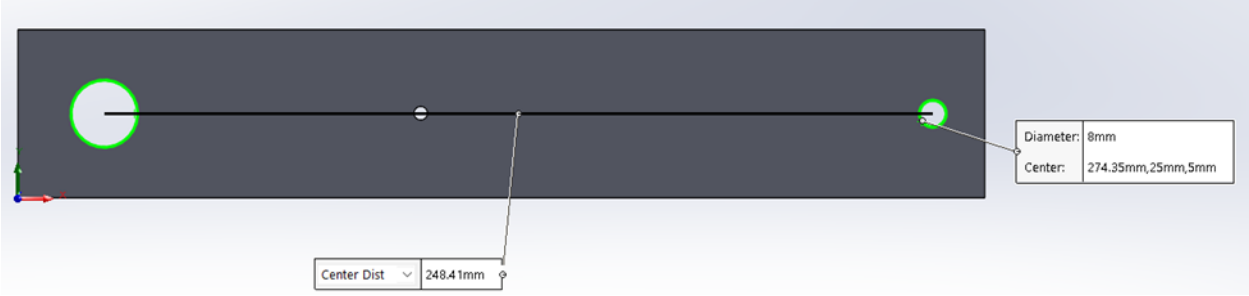


Figure 3: first armhole distance

The second arm is similarly composed of an arm unit that is 210 mm long and 55 mm wide as seen in figure 5. The additional parts consist of an 8 mm bar, bar mount, 40 teeth gear, a timing belt, and ball bearing. The second arm is connected directly to the first arm via an 8 mm bar with a ball bearing, a 40 teeth gear, an 8 mm shaft collar, and a piece of wood. The ball bearing allows for the free and separate motion of the second arm from the first arm. The gear will be attached to a belt that is driven by the Nema 17 motor attached to the first joint. The 8 mm shaft collar and 8mm rail guide support are physically attached to the 8mm bar and will act as the point of connection to the second arm. The arm will have bolts driven through it such that they connect with the rail support and guide the rail. The end of the arm will serve as a mount for the gripper. The base design of the component can be seen in figure 4.

Figure 4: second arm design

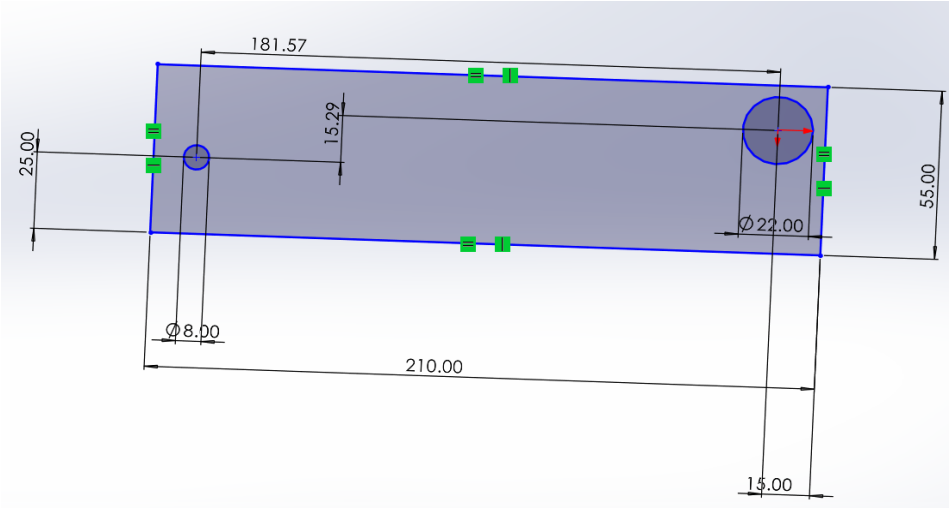


Figure 5: second arm measurements

Grabber

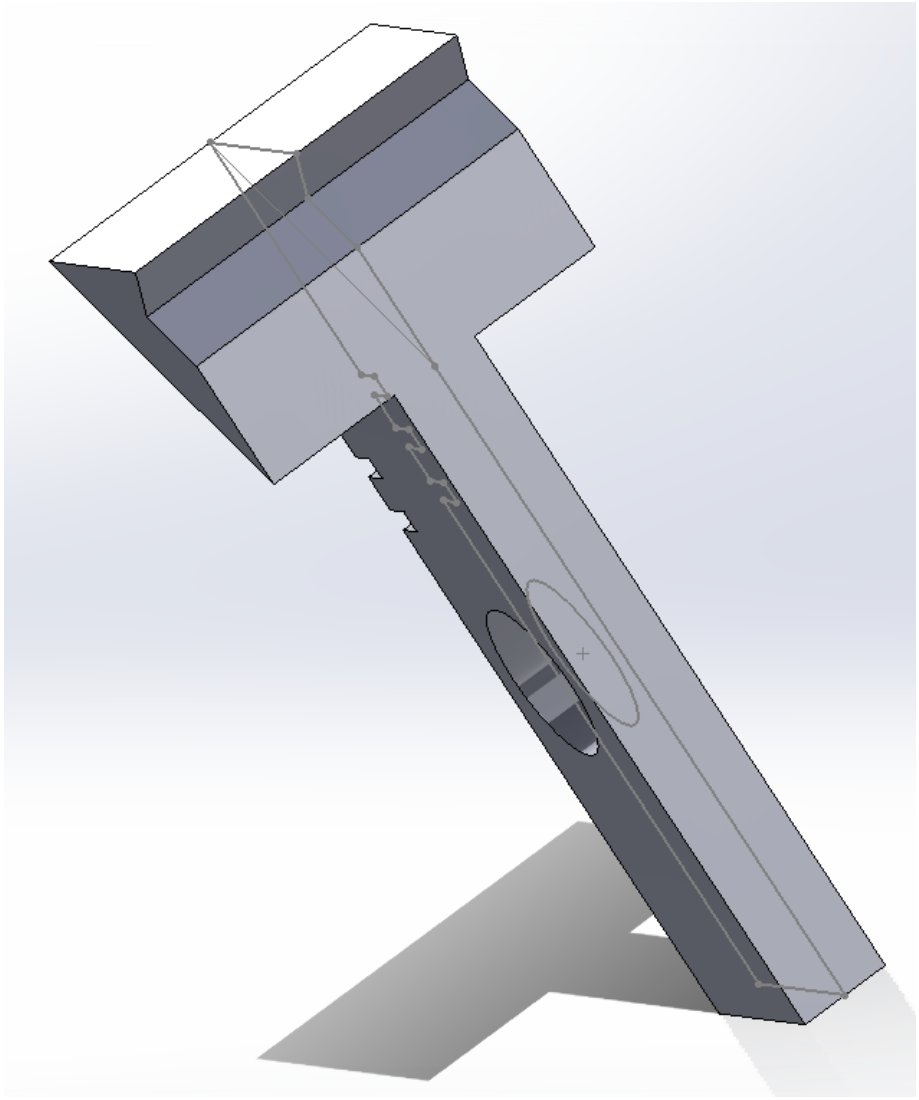


Figure 7: Grabber part 1

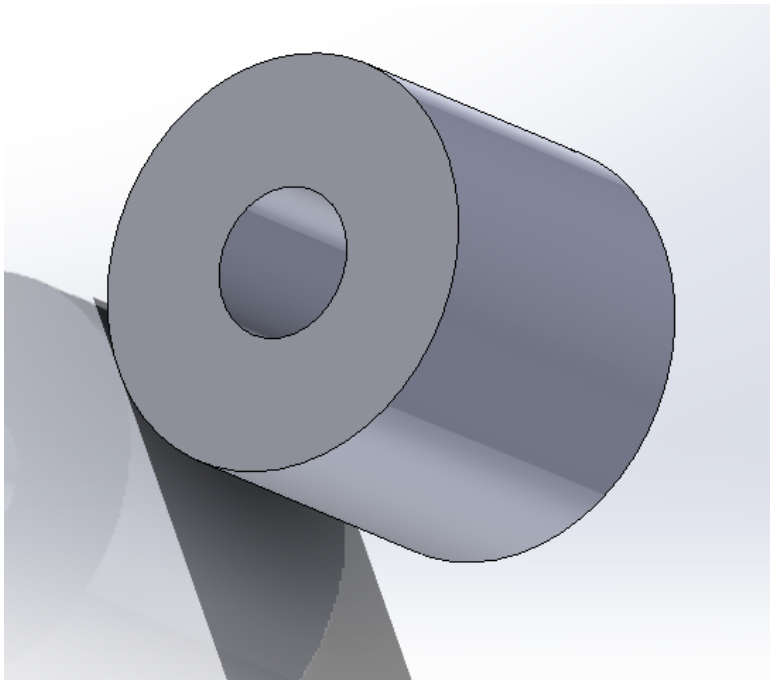


Figure 2: Grabber part 2

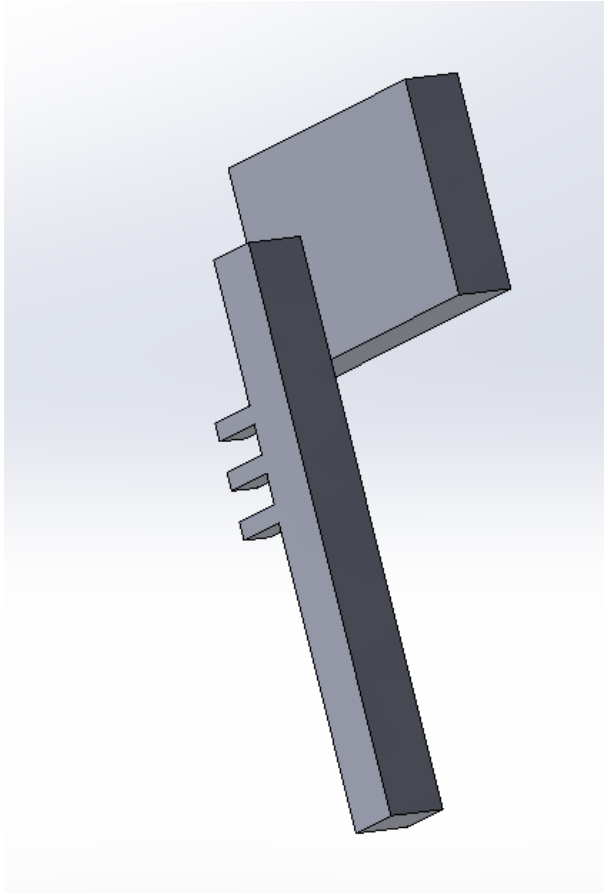


Figure 3: Grabber part 3

The grabber is a 3d-printed, completely mechanical design with zero electronic components. There are a total of three parts in the grabber. The first part is the main rotating grabber part, which is mounted on a bearing in the second arm. The slots are designed for rubber bands to go through them to attach to the third grabber part. The second part is a spacer that was pressed into the hole inside the main rotating grabber part to reduce friction and give it a bit of height. The third part is a tensioner and wall, which is designed to create optimal points of contact to hold the tool, and it has pegs to hook the rubber bands mentioned previously, to maintain tension and grip the tool properly. It is designed to easily release and hold a tool by moving the back part of the main tool right (when looking forward from the perspective of the second arm) and releasing it to secure the new tool.

Enclosure

The enclosure is shaped as a square box with a height of 70 mm see figure 6. This height is selected such that the enclosure can mount the Microcontroller and the PCB, as seen in figure 7. The Microcontroller has a length of 53.34 mm and a width of 68.58mm. At the same time, PCB is a rectangle with a length of 48mm and a width of 27mm. The width and length of the enclosure are 140 mm see figure 5. The more extensive base prevents tipping and allows ample room to enclose the motor and an arm mount. However, a bottom panel will not be added to allow easy access to the contents within the enclosure. The enclosure is made out of 5 mm thick plywood. The microcontroller and PCB are mounted on the outside to allow for easy movement of wires and wiring of motors.

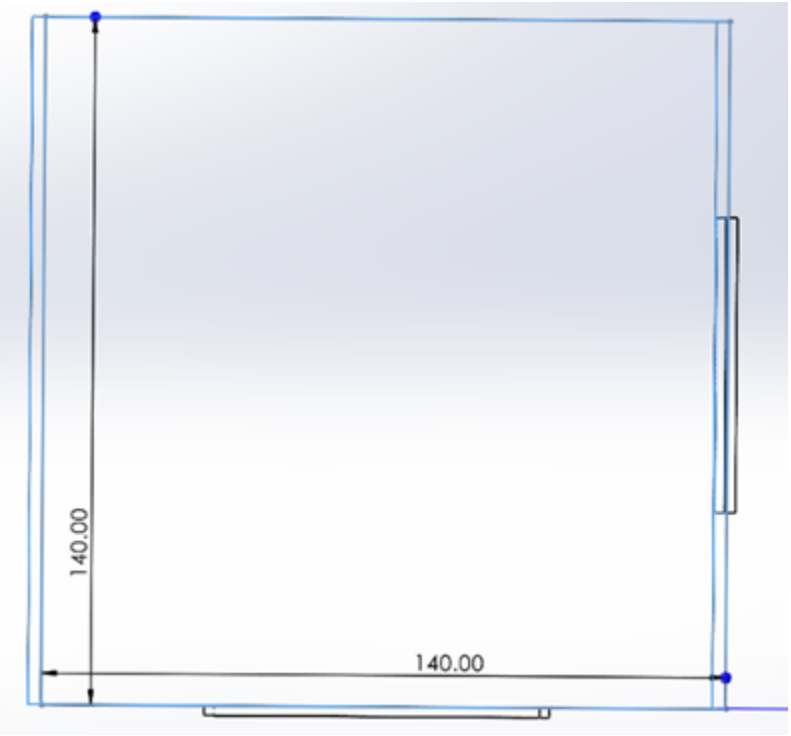


Figure 6: Top-down view including dimensions

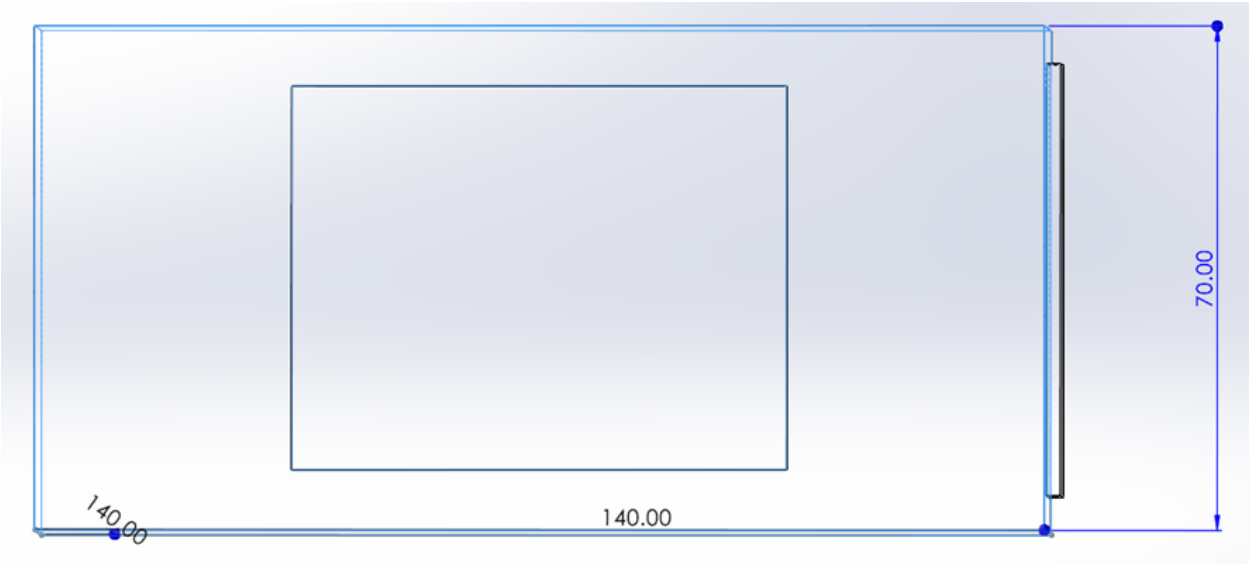


Figure 7: Front-facing view showing height in millimeters.

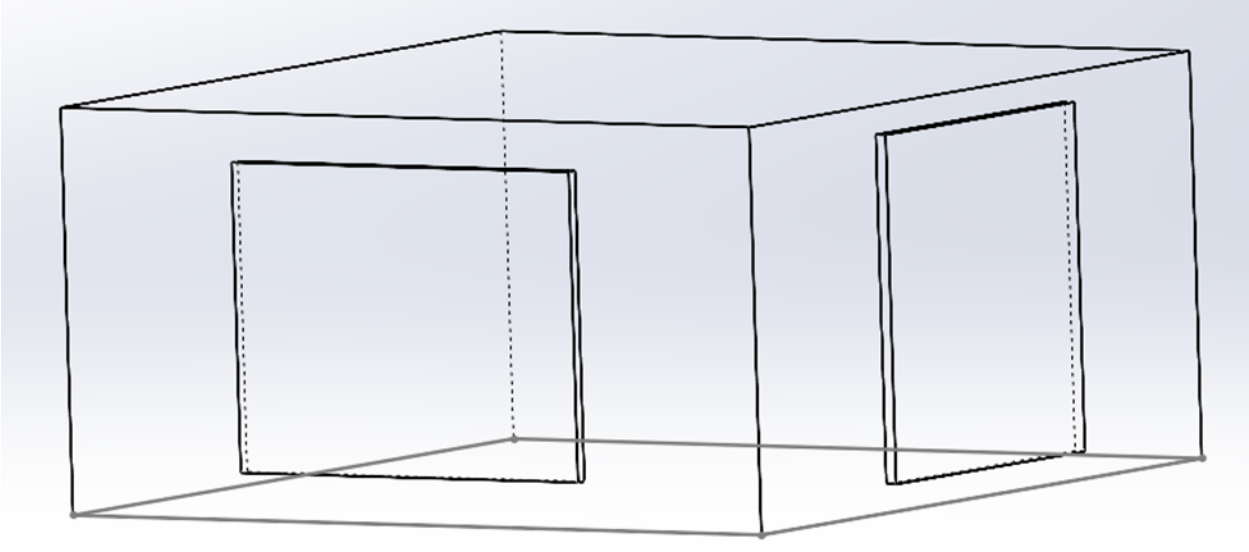


Figure 8: Full enclosure

PCB

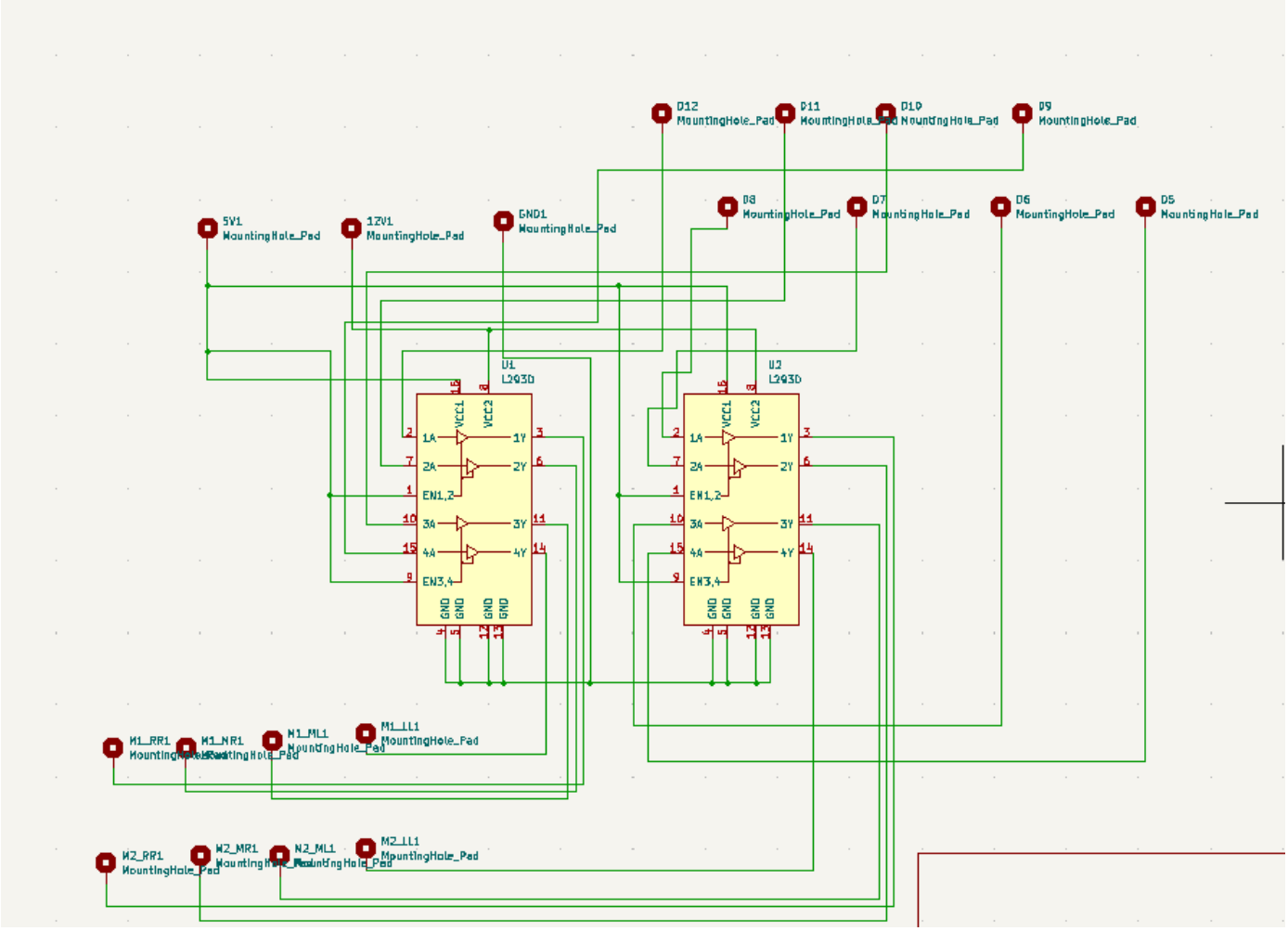


Figure 9: PCB Schematic

Note:
Both RR1 pins connects to NEMA 17 pin 3 of respective motors
Both MR1 pins connects to NEMA 17 pin 4 of respective motors
Both ML1 pins connects to NEMA 17 pin 2 of respective motors
Both LL1 pins connects to NEMA 17 pin 1 of respective motors
For NEMA 17 pin descriptions, see the datasheet links in this document

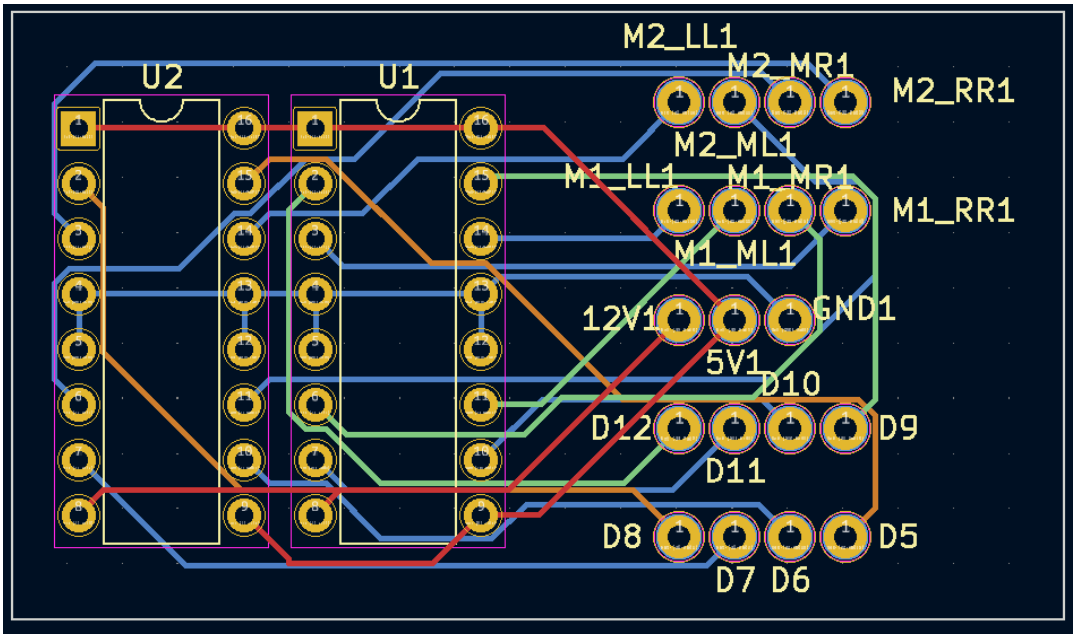


Figure 10: PCB Routing Diagram

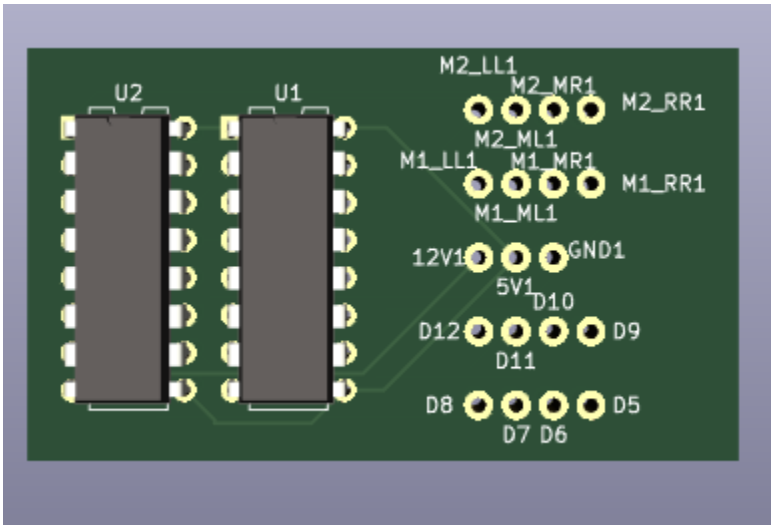


Figure 11: PCB 3D view

The PCB was ordered from OSHPark. It is 48.26 x 27.94 mm. It has 2 L293D ICs on it; the left one (U2) controls motor 2, and the right one (U1) controls motor 1. It has 5 rows of vias; the first row is the top row, and the last row is the bottom row. Standard male pin headers were soldered into the vias for easy and convenient jumper wire implementation. The first row consists of 4 holes, the outputs going to the second motor. The second row consists of 4 holes, the outputs going to the first motor. The third row consists of 3 holes: the 12V power input, the 5V power input, and GND. The fourth row consists of 4 holes, the control inputs for motor 1. The fifth row consists of 4 holes, the control inputs for motor 2.

Datasheet Links

NEMA 17: E:\小狄\(1)CAD图纸\MybotOnline\17HS15-1704S Model (1)
L293D: L293x Quadruple Half-H Drivers datasheet (Rev. D) (ti.com)

Bill of Materials

	Distributor	Price	Purchaser	Total:
Arduino Uno copy	Resis-store	$6.00	David	$168.86
USB - A cable	Resis-store	$2.00	David	
NEMA-17	Tekbots	$10.00	David	
L293DNE	Tekbots	$4.00	David	
NEMA-17	Tekbots	$10.00	David	
L293DNE	Tekbots	$4.00	Nishant	
Plywood	Homedepot	$10.98	David	
2xelbow brackets	Homedepot	$10.94	David	
5xmachine screw bags $1.38 per bag	Homedepot	$6.90	David	
3xPCB	Homedepot	$20.90	Nishant	
acyrilic arm 1	Tekbots	$0.00	David	Voucher used
acyrilic arm 2	Tekbots	$0.00	David	Voucher used
5x 5 mm 20 tooth gear	Amazon	$6.99	David	
2x Stepper motor L brackets	Amazon	$8.99	David	
2x 8 mm diameter, 100 mm long steel rod	Amazon	$7.99	David	
2x 8mm rail clamp support	Amazon	$4.58	David	
5m GT2 timing belt 6mm width	Amazon	$8.99	David	
5x 8mm shaft collar	Amazon	$6.49	David	
2x 8mm 40 tooth gear	Amazon	$7.99	David	
10 x 8mm steel ball bearing	Amazon	$7.50	David	
Super glue	Homedepot	$4.67	David	
Jumper cables female to female/ male to male	Tekbots	$12.00	Nishant	
M8 bolt	Homedepot	$4.75	David	
Mini spring clamp	Homedepot	$0.60	David	
8 oz mason jar	Costal	$1.60	David	
2x 3d print	Tekbots	$0.00	Nishant	Voucher used

Time Report

Group Time:

Weekly meetings: 18 hours
Lab time: 18 hours
Paperwork: 3 hours
Last minute sprint: 20 hours

David Jepsen: Total time spent on the project (including group time): 44 hours

Hours spent on Enclosure design: 4
hours spent on Enclosure construction: 6
hours spent on Arm design: 7 hours
Hours spent on Arm construction: 5 hours
Hours spent on paperwork: 4 hours

Logan Kesting: Total time spent (including group time): 98 Hours

Learning Python/Constructing GUI: 10 hours
Learning/implementing Inverse Kinematics in Python: 4 Hours
Implementing Arduino/Python Serial Communication: 20 Hours
Implementing AccelStepper.h (Arduino motor driving code): 5 Hours

Nishant Sane: Total time spent (including group time) : 103 Hours

Learning how to calculate inverse kinematics and speed: 6 hours
Learning how to use MultiStepper.h to simultaneously control motors: 20 hours
Implementing variable-controlled procedural Arduino script: 8 hours
Designing PCB: 10 hours