# EE175AB Final Report Template
# Project Title

## EE 175AB Final Report

## Department of Electrical Engineering, UC Riverside

| | |
| --- | --- |
| **Project Team Member(s)** | Maverick Bautista, Edward Haro, Antonio Garcia |
| **Date Submitted** | 06/13/2023 |
| **Section Professor** | Roman Chomko |
| **Revision** | Revision 4.0 (06/13/2023) |
| **URL of Project YouTube Videos, Wiki/Webpage** | Final Output:<br><br>https://youtu.be/0tLkIu9rlb4<br><br>Image Conversion to G-Code Conversion:<br><br>https://youtu.be/n28ljvZNEEk |
| **Permanent Emails of all team members** | mbaut030@ucr.edu<br><br>eharo010@ucr.edu<br><br>agarc541@ucr.edu |

**Summary**

This report presents the work contributed to the Handwriting Copying Machine for EE175AB senior design.

# Revisions

| Version | Description of Version | Author(s) | Date Completed | Approval |
| --- | --- | --- | --- | --- |
| 1.0 | First draft with inclusion of 1.0 Executive Summary, 2.0 Introduction, and 3.0 Design Considerations | Maverick Bautista, Antonio Garcia, Edward Haro | 01/29/2023 | |
| 1.1 | Addition of 4.0 Experiment Design and Feasibility Study, 5.0 Architecture and High Level Design, and 6.0 Data Structures.<br>Edits to 2.0 Introduction and 3.0 Design Considerations. | Maverick Bautista, Antonio Garcia, Edward Haro | 02/05/2023 | |
| 1.2 | Addition of 7.0 Low Level Design.<br>Edits to 3.0 Design Considerations, 4.0 Experiment Design and Feasibility Study, 5.0 Architecture and High Level Design, and 6.0 Data Structures. | Maverick Bautista, Antonio Garcia, Edward Haro | 02/12/2023 | |
| 2.0 | Addition of 8.0 Technical Problem Solving, 10.0 Test Plan, and 11.0 Test Report.<br>Edits to 5.0 Architecture and High Level Design, 6.0 Data Structures, and 7.0 Low Level Design. | Maverick Bautista, Antonio Garcia, Edward Haro | 02/26/2023 | |
| 3.0 | Addition of 12.0 Conclusion and Future Work, 13.0 References, and 14.0 Appendices.<br>Edits to 8.0 Technical Problem Solving, 10.0 Test Plan, and 11.0 Test Report | Maverick Bautista, Antonio Garcia, Edward Haro | 03/21/2023 | |
| 4.0 | Final draft with edits to all sections | Maverick Bautista, Antonio Garcia, Edward Haro | 06/13/2023 | |

# Table of Contents

# 1 Executive Summary

The Handwriting Copying Machine is a device that is used to capture and reproduce images. The motivation is to design an accurate plotter with speed. The team set out to create a complex project and landed on the idea of creating a pen plotter. It is an XY-axis plotter that moves a utensil held in place. The plotter is 210 x 297 mm where two stepper motors move a belt system. A servo motor attached to the end of the arm vertically moves the utensil. Ideally, the plotter replicates the image used for input in a manner that resembles human handwriting and drawing. The project is split into three main objectives, split between the three team members: image to SVG conversion, SVG to G-Code conversion, and G-Code to motor commands. It highlights the use of drawing for artists and graphic designers alike to share their work. Finally, data can be analyzed to understand the complex structures of raster images and binary pixels.

Operation begins with a Python script using the OpenCV library. GUI features are used to instruct the user to select an image file. Afterwards, the image is converted into binary coordinates stored and processed into contours. The detected contours are converted into paths on an SVG file. The next step is to parse the SVG file and create a G-Code file. In the new file, commands are instructed or placed depending on the coordinates given. Finally the microcontroller is reprogrammed to read in G-Code.

Integration between SVG and G-Code conversions allows for any image to be converted to an SVG and G-Code file within the same script. The designed pen plotter is then able to take an inputted image on a computer and is able to recreate a pixel based image version of the input to be drawn on an A4 paper. The image created by the system is a skeletonized monocolor replication of the original imputed image.

Contour detection is needed to convert it to SVG. Being able to recognize and distinguish paths from the resulting SVG file is essential to the creation of the G-Code, especially for the lifting and lowering of the pen on the plotter itself. The pen plotter tests were successful and continued to improve as the project progressed. The original mechanical system had a prebuilt PCB which had to be reverse engineered to find the configurations of pins used on the MCU for reprogramming purposes. The mechanical system is then programmed to recreate a rough sketch similar to the original image. The system is programmed to be within a 0.5 mm range of the G-Code input in order to preserve a combination of high speed motor control and accuracy.

Although the end result is not an exact replication of the original image, each objective is competent enough to create an output that clearly resembles the original image. Due to the way the G-Code file is inputted into the PIC18F, the G-Code files are specially designed for this project and are not compatible with other plotters. The mechanical system is based on the iDraw 1.0 plotter; it can be modified to fit the needs of a CNC machine or into a laser printer.

# 2 Introduction

## 2.1 Design Objectives and System Overview

The Handwriting Copying Machine is an XY-plotter that uses computer vision to “see” a 2D image and replicates it with the attached utensil. The goal of the project is to have a design that can copy images and drawings in a way that resembles human handwriting. The project is meaningful as it applies prior knowledge of electrical engineering principles and challenges each group member with their own specialized task. This project is important as it incorporates computer vision software to conduct embedded motor movements that interfaces between a PC and microcontroller. It also highlights the visual aspect of the work conducted to enable hardware and software. Likewise it combines both digital and physical drawings to be produced. The design uses the knowledge of embedded system programming, data acquisition, computer vision, and serial communication to have a functional project.

An SVG is created by reading the contours from the image using computer vision. The image is then converted to binary and filtered using thresholding. The points that were obtained are listed into an array that retrieves the paths to each point. The array of points is then converted to an SVG file.

The file is then opened from the folder and read through to find the paths using the XML library that comes with Python. After the paths are placed into nested lists, the coordinate points are converted from pixels to millimeters for proper output. From here, each coordinate is added into a newly created G-Code file, where G commands and X and Y lettering are appended to it. After the homing command is appended, the file is closed and is communicated to the microcontroller.

Communication of the microcontroller runs through a USB input that connects to the PCB with UART interface. The connected computer would send the G-code commands to the microcontroller and read them in until it needs to stop reading in commands. Once processed, the motors move to the corresponding coordinate distance in millimeters starting from the origin starting point.

Motors move by having the PIC18F microcontroller reprogrammed to have the new G-Code array with the PICkit 4.0. Reflashing commands the motors to draw something different compared to the first drawing. Then the G-code gets read and changes the millimeter set points from the array into steps for the stepper motors and moves the utensil to the set location.

The primary objective is to efficiently and effectively plot precise sketches that can be completed in a few seconds to minutes with precision of 1 mm accuracy. Image processing is limited by the camera used to take the image and the end product is to be drawn on A4 paper. The captured image must be less than 50 Kilobytes in size to prevent memory overload.

The project is separated into 3 main parts by the members: Maverick Bautista converts the raster images into SVG, Edward Haro converts the SVG into G-Code, and Antonio Garcia flashes the PIC controller to conduct motor movements.

## 2.2 Backgrounds and Prior Art

Checks were made to see if the project is a known invention.

**Ender 3** - The Ender 3 is a popular 3D printer that is priced at around $189.99 on Amazon and from the direct vendor [1]. For 3D printing, the Ender is exceptional at the price and its ability to be modified into a CNC machine. However, it utilizes a bed that is independently moved instead of the normal XY-axis of a plotter and the nozzle will need to be modified into a utensil holder.

**AxiDraw** - A simple XY-plotter that utilizes 2 stepper motors for movement and a servo for the Z-axis. There is comprehensive performance, physical, and software information listed on the site to help users [2]. However, the biggest issue with this plotter is that the cost is $550.00 to purchase.

**iDraw** - A similar design to the AxiDraw but at a reduced cost for $250.00. The website has much more information when compared to the AxiDraw with images and video tutorials [3]. The PCB and specifications are also provided to help users who want to modify their own plotter. As a result, this is the plotter the team chose to house the hardware and frame. The PCB used by the iDraw is a modified version of the open source EBB board.

**RobotHouse** - The overall design is similar to the AxiDraw and iDraw in its structure and design implementation [4]. However it costs slightly more and there is a lack of resources regarding the product.

**RepRap** - An open sourced site that documents many DIY 3D printers [5]. There are many projects that can be chosen. However, as a 3D printing site, most of the projects are 3D printers meaning modifications will need to be made. Alongside, many of the projects have a lack of resources or instructions to follow when building. Another factor was the cost as some are estimates with many requiring access to a 3D printer to 3D print some components or lack of availability online.

**HandWriting Copying Machine** - The design has less accuracy and is more complex to start when compared to similar existing products as it is built upon the iDraw. The product's advantage is in the user's ability to input a picture captured.

## 2.3 Development Environment and Tools

**Hardware**

The system is built upon the iDraw 1.0 pen plotter. The plotter includes a metal frame, belt system, 2 stepper motors, a servo motor, and the iDraw board. A PICkit 4.0 is used to reflash the iDraw board. A soldering kit is used to solder some of the pins for connections between the PICkit and iDraw board.

**Software**

Jupyter Notebook is used to integrate the OpenCV library using the Python language. The code is available on GitHub for any user to have access for cloning or editing the code. For the G-Code conversion, the code was implemented and tested using PyCharm. After verifying that G-Code commands were done correctly, it is integrated into the Jupyter Notebook. To program the EBB board, the software needed is MPLAB X IDE that is programmed in using C.

**Testing Equipment**

To verify pin inputs and outputs from the PCB, a multimeter is used to test connections. Alongside determining the voltage levels to ensure the correct voltage is being distributed. An Arduino is used to verify UART communication within the PC system. Using a breadboard, it allowed for safe and consistent programming of the iDraw board.

## 2.4 Related Documents and Supporting Materials

Several resources were used for research and feasibility development. The IEEE 754 is an industry standard for Python floating point which is implemented into the Python script. The PIC18F datasheet is needed to determine the pin connections to the iDraw board [6]. The MPLAB site provided the software installation and guidance to reprogram the iDraw board. A UART implementation for PIC microcontroller was reviewed for serial communication [7].

## 2.5 Definitions and Acronyms

Two separate tables are listed with Figure 2.5.1 for definitions and 2.5.2 for acronyms.

**Definitions**

| Definition | Explanation |
| --- | --- |
| EBB Board | EiBotBoard is a USB port driven board that is focused on dual stepper motor control. |
| iDraw Board | Modified variation of EBB board. |
| G-Code | Instructions given to a machine controller that instructs motor movement, speed, and pathing. |
| MCU | Microcontroller unit is a small microcomputer embedded on an integrated circuit (IC) chip. |
| OpenCV | An open source computer vision software library. |
| Raster | A two-dimensional plane represented by computer graphics represented by a square pixel grid layout. |
| SVG | Standard Vector Graphics are computer graphics constructed using geometric shapes defined on the cartesian plane. |
| UART | Universal Asynchronous Receiver/Transmitter enables communication with configurable speed. |
| USB | Universal Serial Bus enables communication between devices and the host controller. |

**Figure 2.5.1** List of definitions.

**Acronyms**

| Acronym | Meaning |
| --- | --- |
| CNC | Computer Numerical Control |
| CV | Computer Vision |
| DC | Direct Current |
| DIY | Do it yourself |
| GUI | Graphical User Interface |
| PC | Personal computer |
| PCB | Printed Circuit Board |
| XML | Extreme Markup Language |

**Figure 2.5.2** List of acronyms

# 3 Design Considerations

## 3.1 Realistic Constraints

There are 6 major constraints related to the hardware, 3 related to software, and 2 related to power consumption. The constraints are listed in order of importance:

- Total project cost is $361.71

The hardware constraints are:

- Single color sketch based on utensil used should be ball-point tipped pen or a marker
- The environment is in a well lit location.
- The camera used is from a group-members phone.
- The paper is a blank (white) A4 size.
- Sketch completion duration varies according to image size and complexity.

The software constraints are:

- The PC System needs the IDE to run the OpenCV library.
- The converted file cannot be greater than 50 Kilobyte.
- The camera picture resolution will be set to 2480 x 3508 pixels (A4 size)

The power consumption constraints are:

- 12 V AC/DC 1 A wall adapter
- 8 MHz MCU Clock

## 3.2 System Environment and External Interfaces

The system will operate on a PC system that any member has access to. The IDE environment is Jupyter Notebook. The notebook runs the OpenCV library in Python. The MCU is a part of the PIC18F family of microcontrollers. The PIC microcontroller is flashed and coded with C using MPLAB X IDE. Through a USB port, the MCU is connected to the iDraw board and PC to allow communication using the PICkit 4.0. Following that, the microcontroller reads in G-Code commands to allow motor functions [8]. The XY-axis is moved by 2 stepper motors and the Z-axis is moved by a servo motor.

## 3.3 Industry Standards

All of the standards mentioned in Figure 3.3.1 are industry software specifications used in the design. The standards comply with the Jupyter Notebook and MPLAB X environments. The only standard that affected the hardware is the IEEE 754. The arithmetic calculations from Jupyter Notebook are rounded to the nearest 0.16 mm in MPLAB X. Refer to Appendix A for the referenced documents and materials for standards.

| Industry Standard | Explanation |
| --- | --- |
| IEEE 754 | The technical standard for floating point arithmetic within Python |
| IEC 61508 | Allows testing and verification for safety systems between electronics and software |
| IEC 62304 | Safety standards within embedded software |
| IEC60730 | Provides the tests and diagnostic methods for operations |
| RS-232 | A standard communication protocol when connecting a PC to peripheral devices using serial data transfer |
| USB | Universal Serial Bus enables communication between devices and the host controller |

**Figure 3.3.1** Industry Standard Table

## 3.4 Knowledge and Skills

All members have experience from previously attended courses. The members have also gained knowledge and skills throughout the creation of the project.

**Maverick Bautista**

**Prior knowledge and skills:**

Introduction to Computer Science I CS10A, Introduction to Computer Science II CS10B, Introduction to Embedded Systems EE/CS120B, Sensing and Activation for Embedded Systems EE128, Soldering

**New knowledge and skills:**

Python Programming, Computer Vision, Graphical User Interfaces, Communication

**Edward Haro**

**Prior knowledge and skills:**

Introduction to Computer Science I CS10A, Introduction to Computer Science II CS10B,

Introduction to Embedded Systems EE/CS120B, C Programming

**New knowledge and skills:**

Python Programming, G-Code, Sensing and Activation for Embedded Systems EE128

UART Communication

**Antonio Garcia**

**Prior knowledge and skills:**

Introduction to Computer Science I CS10A, Introduction to Computer Science II CS10B, Sensing and Activation for Embedded Systems EE128 Motor Controls, Sensing and Activation for

Embedded Systems EE128 Arduino Programming, Introduction to Embedded Systems EE/CS120B, C Programming

**New knowledge and skills:**

Microcontroller Flashing, MPLAB X IDE, Sensing and Activation for Embedded Systems EE128 UART Communication

## 3.5 Budget and Cost Analysis

In the table below, Figure 3.5.1 lists the hardware components purchased and the total cost of the project. There are other parts included that the members previously obtained, refer to Appendix B. The total cost of the project is around $361.71. With it all being for the physical hardware, specifically the iDraw 1.0 plotter. Since the ideal cost is under the $150.00 charge back from the university, the cost is about 2.4 times over budget.

| Item Name | Cost - USD |
| --- | --- |
| iDraw 1.0 Plotter | $250.00 |
| MPLAB PICkit | $88.54 |
| Arduino CNC Shield | $7.99 |
| Motor Drivers | $10.19 |
| Limit Switches | $6.99 |
| | Total Cost: $361.71 |

**Figure 3.5.1** Cost of items

## 3.6 Safety

There are a few safety concerns for the project in a few aspects of power delivery and physical wellbeing. The system is powered by a DC 12V 1A power adapter that is connected to the PIC microcontroller. The PIC is connected to a set of resistors on a breadboard and to the PICkit for consistent programming purposes. The PICkit is connected with a 5V USB cable to power itself. It is recommended to not have contact with the resistors during the reprogramming process as it can cause a failure during the process and can cause harm with direct contact. Finally there will be a light source needed for precise and clear photos to be taken. Individuals with eye-sight problems must stay clear of the light source.

## 3.7 Performance, Security, Quality, Reliability, Aesthetics etc.

The machine accomplishes its objective in creating a sketch of the inputted image within 0.16 mm precision. The machine makes the sketch at a speed where noise limitation was a main priority as increasing the speed made the machine significantly louder. To ensure the paper does not move freely, a clip board is used to attach the paper.

For image quality, a light source prevents shadows from being accidentally identified as contours. The image captured can only be as reliable as the camera being used and the environment it is in.

## 3.8 Documentation

Documentation is kept on Confluence for key information and links to important resources. A timeline is included to monitor progress. Each member also had a dedicated notebook to write down any feedback or experimentation. Diagrams and circuit sketches were constructed using diagram.io within Google Drive for members to access and edit documents. Any other related documentation was shared on Google Drive. Code is uploaded to the specific members' GitHub repositories, refer to Appendix C.

## 3.9 Risks and Volatile Areas

One major risk involves computer vision with the implementation of the contours detected. A computer vision error occurs if the user were to draw something completely filled in. The issue is with the method used for morphology and erosion which causes the image to look entirely different. One such example is drawing a big filled in square, the detected contours will be a large X as the paths for skeletonization revolve around the average taken from the parallel contours to generate a new contour. The skeletonized contours created overlap very closely within the 0.16 mm which often causes the machine to overlap where it previously sketched for the same image.

# 4 Experiment Design and Feasibility Study

## 4.1 Experiment Design

There are three major experiments to test the feasibility of the Handwriting Copying Machine. Contour detection, communication between the PC and MCU, and reverse engineering the MCU connections on the PCB. Refer to Appendix D for equipment used for testing.

### 4.1.1 Contour Detection

**Objective:** Find and detect the contours of an image that will be visually shown to the user.

**Setup:** A Python script is programmed on Jupyter Notebooks IDE. The OpenCV library is installed and imported into the script. The script identifies and detects contours. Afterwards the script places bounding lines over the detected paths.

**Procedure:** Run the IDE with a selected image and follow along the pop-up windows. For this test, an image of multiple shapes is used.

**Expected Results:** The image should be converted and have numerous pop-up windows of the conversion process. It should show an image with its detected contours highlighted in pink lines.

Maverick Bautista was responsible for the computer vision and image processing components. With a PC system using Jupyter Notebooks, a Python script is executed. The script would identify edge detected contours and output multiple GUI windows that visualizes the detection process. The final GUI window will show the image with overlapping pink lines that are the edge detected contours.

### 4.1.2 Communication Between PC and MCU

**Objective:** Establish communication from a PC system to MCU.

**Setup:** Setup a Python script and flash an Arduino using the Arduino IDE that will send a string command.

**Procedure:** Flash the Arduino and ensure it is connected to the PC system via a USB port. Run the Python script with the baud rate set to 9600 [9].

**Expected Results:** The assumption is that the Arduino will read in the data from the USB port and write back a string command as well as echo back the data it received.

Edward Haro was responsible for coding the Python script to send data to the Arduino. The Arduino IDE is installed and coded to test for serial communication. The next responsiblility is the set-up of the experiment with the Arduino and the computer, and for all testing and bug fixes required for the experiment.

### 4.1.3 Determining Pins

**Objective:** Reverse engineer the MCU connections on the PCB.

**Setup:** Connect a multimeter to one pin of either the stepper motor and then to the MCU for a continuity test.

**Procedure:** Connect multimeter to each pin until continuity test confirms a connection with each motor output on the PCB.

**Expected Results:** Output pins should be identified on the PCB which are connected to the motor drivers.

Antonio Garcia is responsible for determining the pins of the MCU using the multimeter and for figuring out which pin corresponded to which port.

## 4.2 Experiment Results, Data Analysis and Feasibility

These results and analysis are based on the experiments from section 4.1 in the order they are presented. Based on the results, analysis is provided to assess whether the experiments were achieved and the feasibility of making a project.

### 4.2.1 Contour Detection Results

**Result:** Following the setup from section 4.1.1, the experiment and feasibility test for contour detection is executed. A GUI window named “Detected contours” is the last to appear with pink lines indicating the detected contours as seen in Figure 4.2.1. There are also green bounding boxes stating the pixel length of each identified object.

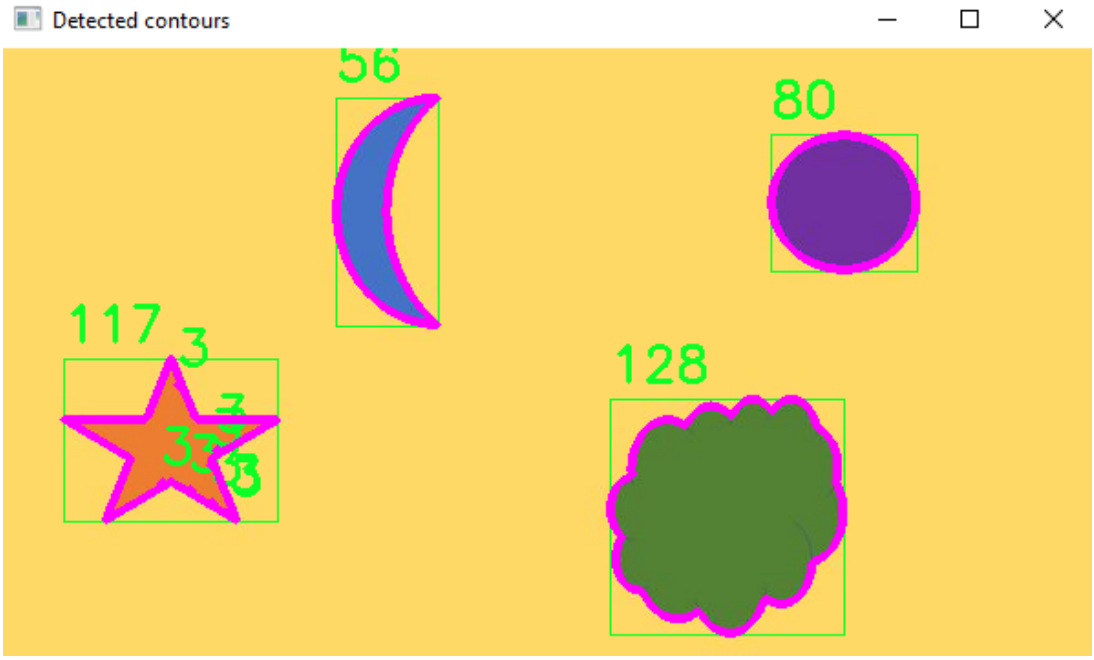


**Figure 4.2.1** Detection of contours highlighted in pink.

**Analysis:** After running the IDE, the edge detected contours are identified and have pink lines placed over each path. This shows that it is possible to detect contours from images and drawings. With this, the first part of the project is shown to be feasible.

### 4.2.2 Communication Between PC and MCU Results

**Result:** The Arduino Uno is flashed with the updated code on the Arduino IDE. The command prompt is checked to see if communication is established between the PC system and Arduino as seen in Figure 4.2.2.


**Figure 4.2.2** Serial Port communication received between PC USB port to Arduino Uno

**Analysis:** Following the procedure from 4.1.2, the command line of the IDE states that the data was received. This shows feasibility for communication, which can be expanded from a simple line to the entire G-Code file.

**Result:** Like section 4.2.2, the Arduino is flashed. The file is read and written to send in G-Code commands.


**Figure 4.2.3** Serial Port communication from PC USB COM3 to Arduino Uno, with Arduino bouncing back what it received.

**Analysis:** Following an expanded procedure from 4.1.2, it is seen that it is in fact possible to send over an entire G-Code file line by line. This is needed for the MCU to successfully read in the file. Because of the success of the experiment, this part of the project is shown to be feasible and shows that this technical design objective of communication can be achieved at least on one end.

### 4.2.3 Determining Pins Results

**Result:** A total of 7 ports for input and output were identified and used to conduct motor movement after the procedure from 4.1.3.

**Analysis:** Once each of the pins were determined on the EBB board, work can be done to reprogram the MCU. With each pin and port identified, it is now possible to reprogram the MCU and the feasibility test is successful.

Based on the feasibility results, it was determined that each test was achievable in at least one way. The tests helped choose the best version of the contour detection as well as the most effective way of mapping out the MCU. Communication was also determined as feasible. Once the pins were mapped, reflashing of the MCU was possible and sketches were created.

# 5 Architecture and High Level Design

## 5.1.1 System Architecture and Design

The system architecture and design is divided into 2 main subsystems, the PC System and the Handwriting Robot. The PC system then has image selection and image processing components. The Handwriting Robot contains the power subsystem and MCU. The MCU controls both the stepper motor controller and servo motor controller. Then those subsystems are connected and control the stepper motor drivers. The stepper motor drivers are connected to either of the two X-Y axis stepper motors and the servo motor driver is connected to the Z axis servo motor.

**High Level System Block Diagram**

Figure 5.1.1 displays the architecture of the Handwriting Copying Robot. The system begins with the PC System that runs through Jupyter Notebook. The image selection contains the graphical user interface where it prompts the user to select and input the desired image. Image processing is the next subsystem where the script is executed to detect contours and convert the contour paths into SVG. A contour detection algorithm is performed and is passed into another algorithm to create an SVG file. The SVG file is then parsed and a G-Code conversion algorithm generates a G-Code commands file. The G-Code commands are uploaded to the MPLAB X IDE and compiled to update the Handwriting Robot. Refer to Appendix E for the total software list.

The Handwriting Robot consists of the physical hardware and structural design. The power subsystem consists of the 5 V USB power connection from the PC System to the PICKit 4.0. Alongside a 12 V power supply that connects to the MCU. The MPLAB uploads through the PICkit 4.0 for every instance the compiler is modified. The MCU is the PIC18F microcontroller that connects to 2 stepper motors and 1 servo motor driver. The drivers power 2 Nema 17 stepper motors and an unknown brand of servo motor. The structure of the Handwriting Robot is the iDraw 1.0 A4 configuration. The iDraw 1.0 is an X-Y plotter which came with the MCU and motors.

In Figure 5.1.2, it is a high level circuit diagram that displays the general connections of the iDraw board to other hardware components. The main components needed are the 2 Nema 17 stepper motors and 1 servo motor. A mini-USB connection is from the PICKit 4.0 to USB on the PC System and a 12 V power supply port. Other connections include extensions for extra proprietary connections, pause and reset buttons to restart the board, and a laser connector for a laser attachment.

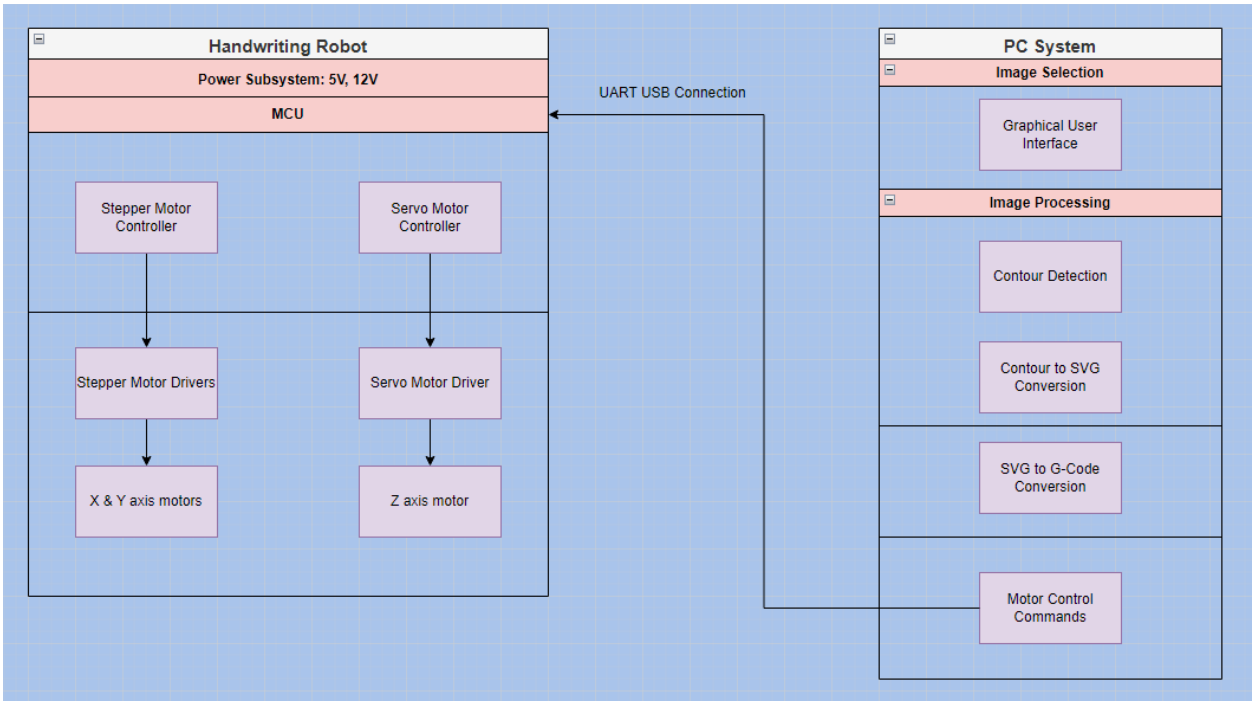


**Figure 5.1.1** High Level System Block Diagram

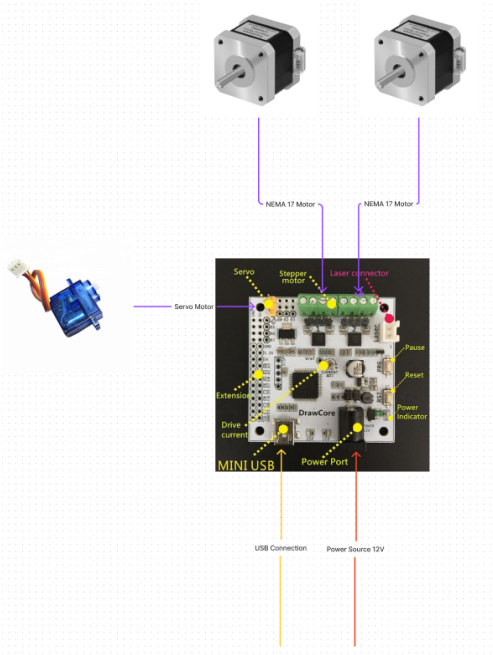


**Figure 5.1.2** High Level Design of Circuit Diagram

Maverick Bautista is responsible for design, debugging, and implementation of OpenCV and conversion of image to SVG.

Edward Haro is responsible for the design, debugging, and implementation of the SVG to G-Code conversion as well as the communication process between the PC and the MCU.

Antonio Garcia is responsible for flashing the MCU which conducts motor control in the Handwriting Robot system.

## 5.2 Hardware Architecture

The hardware system is the Handwriting Robot which has the power subsystem and the MCU as shown in Figure 5.2.1. The power subsystem is a 5 V USB connection to the PICkit 4.0. Alongside a 12 V, 1 A power supply that connects to any wall power outlet. All electrical hardware is located on a PCB called the iDraw board. The iDraw board is a modified EBB board that houses the MCU, power port, mini-USB port, stepper motor connections, servo motor connection, laser connection, drive current, pause and reset, extensions, and power indicators. The MCU is flashed by a mini-USB connection that connects the PICkit 4.0 to the PC System [10]. Once the MCU is programmed, it controls the stepper motor drivers and a servo motor driver. The drivers power 2 Nema 17 stepper motors and a servo motor. Attached to the stepper motors is a belt system. The stepper motors determine the movement along the X and Y axis. While the servo motor moves vertically for the Z-axis.

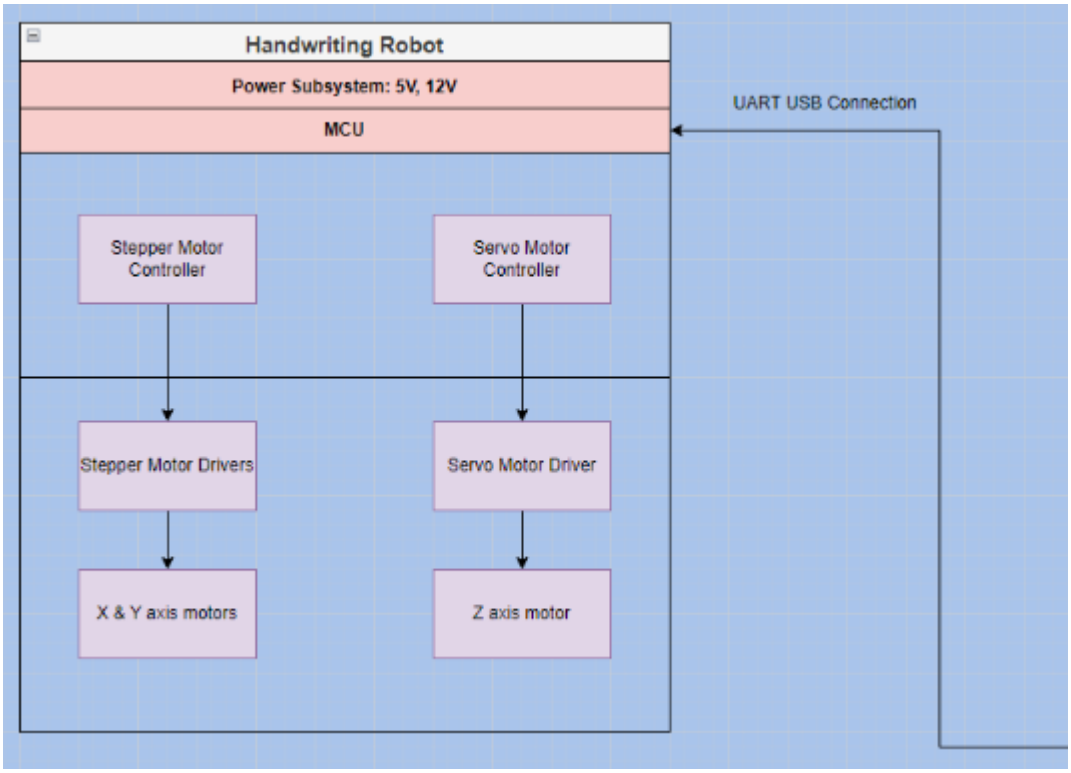


**Figure 5.2.1** Hardware block diagram

Antonio Garcia identified connections on the PCB board without a schematic diagram to implement motor functions by reflashing the MCU which was already on the board. Hardware connections of the system and their components.

## 5.3 Software Architecture

Maverick Bautista is responsible for programming a functional GUI, contour detection, and conversion to SVG. Referencing the PC System diagram of Figure 5.3.1 it uses an IDE to run a Python script. Work was done on Jupyter Notebooks with the OpenCV and TKinter libraries to enable a functioning GUI. An SVG utility and numpy libraries were installed to conduct calculations and conversion to SVG. Using an algorithm, a SVG file is created containing all coordinate paths detected [11].

Edward Haro is responsible for programming SVG to G-Code conversion and communication on a Python script that was implemented after image processing was completed. An XML dom library is installed so that the SVG file created can be read. After the SVG file is parsed, an algorithm is created to modify the text and convert the SVG file into a G-Code command file.

Antonio Garcia is responsible for flashing and reprogramming the MCU using C on the MPLAB X IDE to allow X-Y axis movement using stepper motors and Z axis movement with a servo motor which is seen in Figure 5.3.1. After the pin inputs and outputs are determined, the MCU is reprogrammed. The PICkit 4.0 is required every time the microcontroller needs to be flashed with updated code.

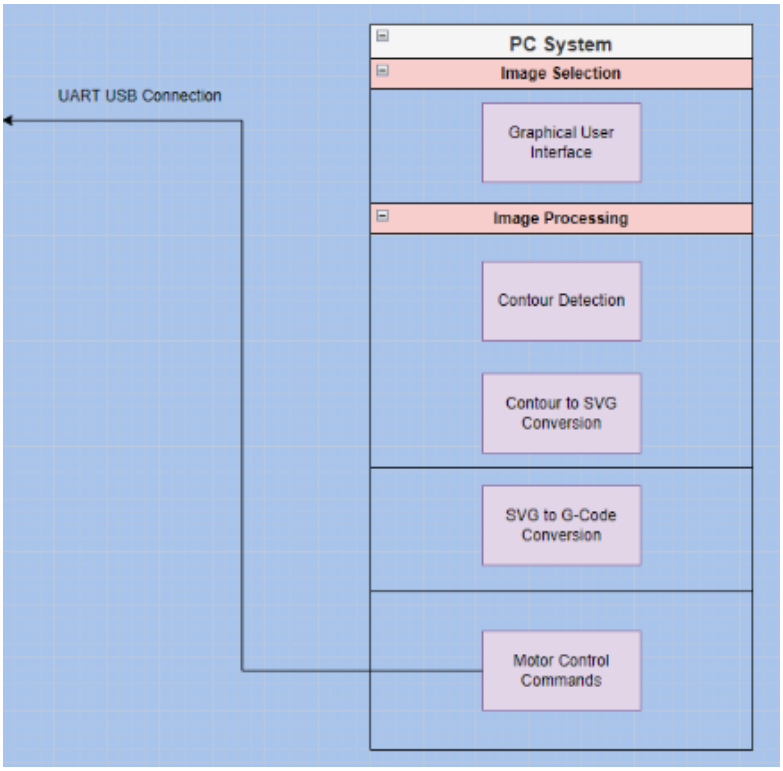


**Figure 5.3.1** Software block diagram

## 5.4 Rationale and Alternatives

Each member had different tasks for feasibility tests and analysis. When researching and experimenting for viability, there were alternatives and reasons for the decisions made.

**Maverick Bautista**

In order to recognize images and their geometric curves, computer vision is needed. OpenCV was chosen as it is an open source and is a free library that all members can download. An alternative for computer vision is to install Nvidia computer vision drivers. The Nvidia drivers required a lengthy process to create an account and acquire the IDE. Unlike OpenCV, the Nvidia drivers can only be programmed using C. Although there is familiarity with C programming, installation of an IDE with Python and OpenCV was a lot easier to set up. Jupyter Notebooks allows flexibility to choose which programming language to code with. For the images taken, they are converted from raster to SVG that can be used for G-Code commands. Alternative methods of plotting allowed the default SVG file to be used as input and be automatically converted to G-Code commands.

**Edward Haro**

Python was used for the conversion and communication due to the ease of integration later down the line as my partner was already using Python, this way no code is needed for major changes when put together. For the approach used for parsing through the SVG file, the XML dom library that comes with Python was used. It allowed for parsing specific areas of the SVG file that were needed for conversion. Where-as before there were a lot of issues with finding the specific line needed, as it is assumed the files are always at the same line. Not every SVG file contains the same amount of paths, so each line must be extracted when parsing a file. After parsing, each path is placed into nested lists to allow for further conversion to millimeters needed for the plotter's MCU. After reading the SVG file and placing paths into their lists, they are converted to G-Code by converting each coordinate and creating a new file that includes the G commands necessary for our applications and appending each coordinate line by line [12].

**Antonio Garcia**

The contingency plan for motor control to reverse engineer the PCB connections for motor control was unable to occur. The plan had an Arduino UNO with a CNC shield and stepper motor drivers connected to move the motors instead of the PCB with stepper motors. The CNC shield is connected with the same 12 V AC/DC adapter that came with the iDraw materials. The Arduino is then programmed using an appropriate C2G USB connection and the Arduino IDE. Refer to Appendix F for the code setup. The Arduino is only powered with the adapter and does not require a PC connection unless the UART connection was unable to be configured properly between the Arduino and the Python IDE. The restrictions for the original plan would be the same except for the 40 K byte size as UART was able to be configured using an Arduino as part of the test phase using UART.

# 6 Data Structures

## 6.1 Internal software data structure

All internal data structures that are passed among components are mentioned here. Components that share memory are between the PIC18F MCU to the PC System.

Antonio Garcia is responsible for an array created to store the G-code into program memory in the MCU instead of SRAM for larger storage capabilities which needs to be reprogrammed in order to change the “drawing” output.

## 6.2 Global data structure

Any data structures that are available that are shared between softwares is seen here.

**Maverick Bautista**

- 2D Array: 'numpy.ndarray'
- Tuple: 'tuple'

**Edward Haro**

- Class ‘list’
- Class ‘str’
- Class ‘int’

**Antonio Garcia**

- ‘const char array’

## 6.3 Temporary data structure

For holding any temporary data structures, files are created for interim use.

**Maverick Bautista**

High level output onto the “path_test.svg” file is created to store the array of path coordinates that is rewritten every time the program runs.

**Edward Haro**

High level output on “svg.gcode” file is created to store the converted svg file to G-code commands that is rewritten every time the program runs.

## 6.4 Database descriptions

N/A

# 7 Low Level Design

## 7.1 PC System

The PC System is divided into two main processes when executed which is shown in Figure 7.1 of a low level system block diagram. The software is performed on a single Python script that runs the SVG and G-Code conversion process. Maverick Bautista is responsible for the SVG conversion. Edward Haro is responsible for the G-Code conversion.

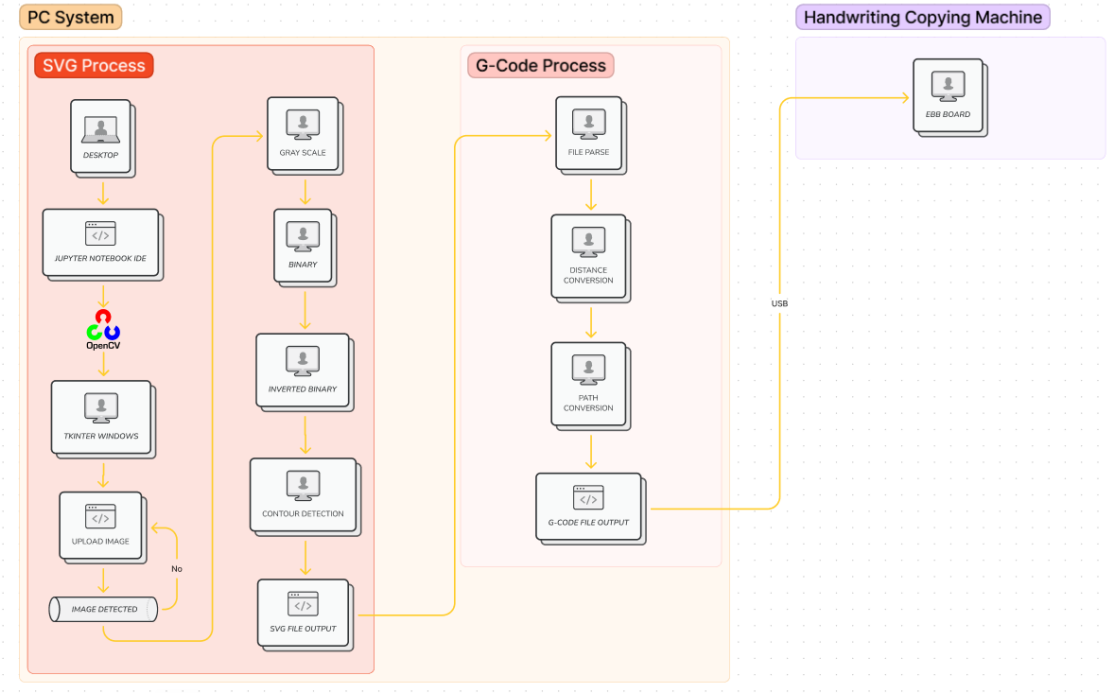


**Figure 7.1** Low Level Design of PC System

### 7.1.1 Processing narrative for PC System

#### SVG Process

The process begins with the user having a PC to use that runs an IDE, in this case Jupyter Notebook. The IDE needs access to run the OpenCV library. Another library called Tkinter is used for GUI functionality [13]. After an image is uploaded, it is checked to ensure that an image is required. Afterwards, image processing is done to the image. It is first converted to grayscale. The grayscale is then converted to binary and finally into inverted binary. Using an algorithm, the inverted binary is analyzed to contour detection. With the contours identified, the values are sent and saved onto an SVG file.

**G-Code Process**

The process begins by opening the SVG file from the destination folder it is placed in; in this case the SVG name and directory never change. Since the SVG and G-Code processes are linked, they are run at the same time and access the same folders. No file or library must be downloaded as all necessary libraries, xml.dom, are downloaded with Python already. Using the XML libraries, the SVG file is parsed and all paths are subsequently placed into nested lists for organization. Coordinates are then converted from pixels to millimeters, and then the path coordinates are used to write the GCode file The process ends with the G-Code output in the same folder and directory as the original SVG file.

### 7.1.2 PC System interface description

**SVG Process**

The first interface a user encounters is a TKinter window. The window asks for the user to select an image. Once that is done, the image goes through several filters where its output is visually seen on the pop-up windows. The first filtered popup is the grayscale conversion. Followed by binary and inverted binary image windows. A final window displays the original image with temporary pink lines that indicate the detected contours. The main output given is the converted paths into an SVG file.

**G-Code Process**

The input is now the aforementioned SVG file that is parsed, manipulated, and converted with the resulting output being the newly generated G-Code file. The G-Code file is sent to the Handwriting Copying Machine module.

### 7.1.3 PC System processing details

A detailed description for each module is presented, including hardware, algorithm, local data structures, design constraints, limitations, performance issues, etc.

**SVG Process**

**Algorithms**

- Guassian blur uses Gausian function within Python to blur an image, allowing more pixel saturation to capture identified pixels and their coordinates.
- Bézier curve applies a parametric curve to smoothen continuous curved lines for, this is applied when a contour is identified based on thresholding to differentiate the binary pixels.
- A skeletonization algorithm is implemented to avoid double contours due to the edge detection, which calculates the average between points, forming a new contour path.
- Converting a contour path onto an SVG file that is from a reference algorithm.

**Data Structures**

- 2D Arrays store the coordinates of each pixel.
- A Tuple stores multiple variables which are the X and Y coordinates within the array.

**Constraints**

- The IDE runs the OpenCV library for image processing and conversion.
- A4 paper size is the set size that fits the physical constraints of the plotter. An A4 paper when converted to a digital plane consists of 2480 x 3508 pixels.

### G-Code Process

### Algorithms

- Pixels to millimeter conversion

A for loop traverses the nested lists and converts each X and Y coordinate in the list to a float to allow for multiplication with the conversion number as it is a decimal. After this, it is converted to millimeters by being multiplied by the conversion number 0.2645. After conversion, depending on where in the list the for loop is in, an X or Y character is placed into the G-Code file and the X or Y coordinate soon follows.

### Data Structures

- Lists - This list is used only to hold all of the nested lists and keep them organized.Nested lists are essentially the SVG paths, and are ordered from first to last.
- Nested Lists - Each nested list is a path extracted from the SVG file. The list begins with the X coordinate, followed by the Y coordinate and so on. The nested list allows for the path coordinates to be accessed and converted. Nested lists also distinguish each path, which is crucial for keeping track of when to move the pen up or down on the Handwriting Copying Machine.

### Constraints

- G-Code output is only for an XY Plotter
- Does not include any input other than the SVG file itself
- To work with our design output must be a single line, for other, more standard designs, the output must have newlines and change the Z axis number.

## 7.2 Handwriting Copying Machine

The Handwriting Copying Machine has the mechanical aspect of the design and includes software aspects in the MCU for the motor controllers. Antonio Garcia is responsible for handling the hardware components of the design. In Figure 7.2.1, it shows the low level system block diagram for the Handwriting Robot. Figure 7.2.2 is the low level circuit diagram that details the connections amongst the electrical components.

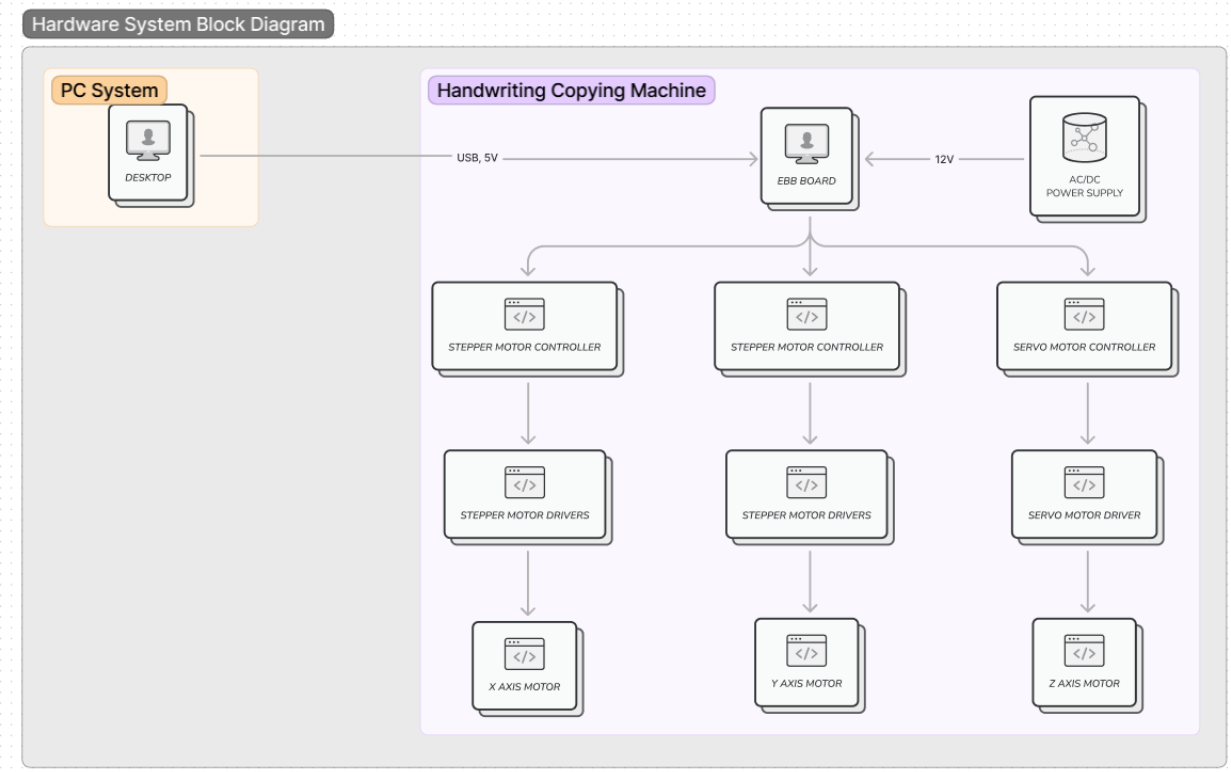


**Figure 7.2.1** Low Level Design of Handwriting Copying Machine

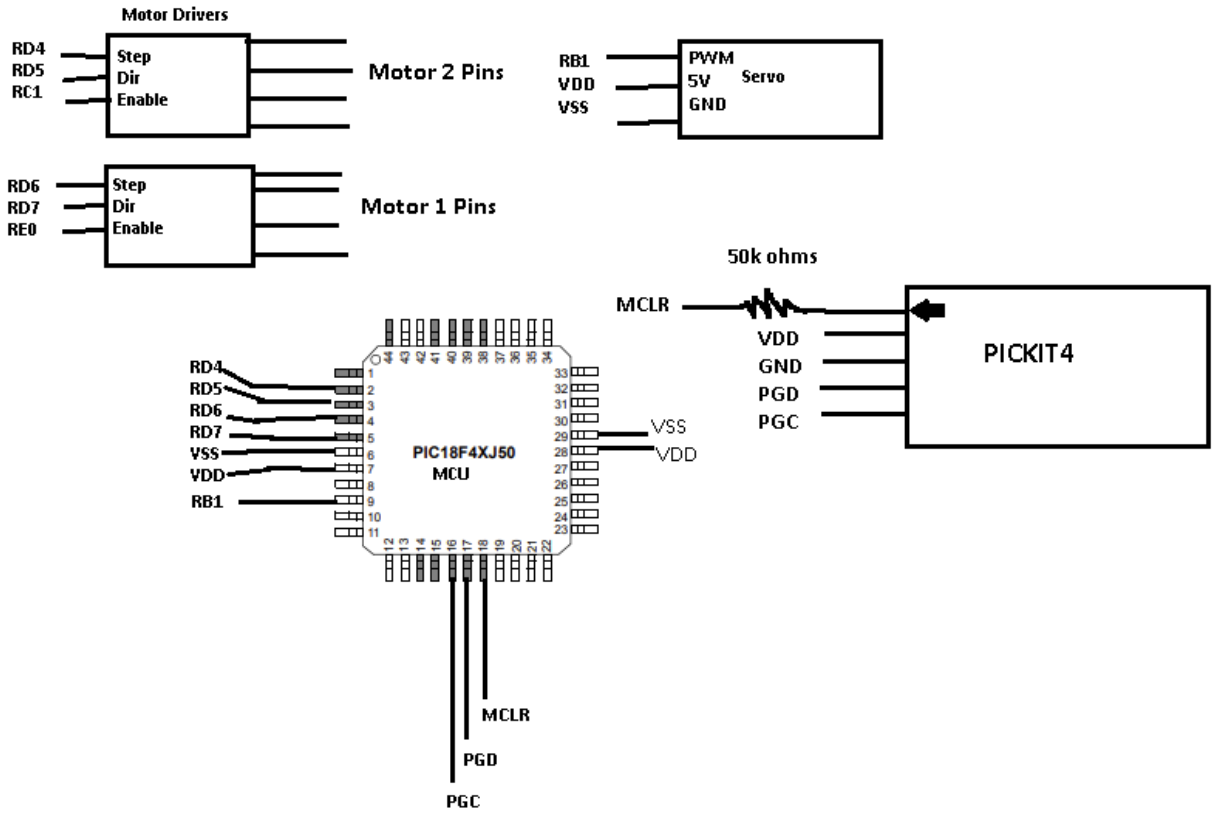


**Figure 7.2.2** Low Level Circuit Diagram

### 7.2.1 Processing narrative for Handwriting Copying Machine

The process begins with the G-code being sent from the PC System module and read into a char array in the microcontroller. Then being converted from mm to steps then being sent as motor commands to the stepper motors and servo motor.

### 7.2.2 Handwriting Copying Machine interface description

The first input is taken in from the PC System module and reprogrammed into the MCU on the PCB. The MCU then runs through the code which reads from the char array filled with G-Code commands and checks which axis its referring too, X/Y/Z, and then runs through and inputs following the chars as integers and shifts and adds them into a holder where it will then be added to the corresponding axis variable [14]. The variable is then changed from mm to steps and processed into the corresponding motors to move the pen.

### 7.2.3 Handwriting Copying Machine processing details

**Algorithms:**

- mm to step conversion
    - mm input / (32/200)

**Data Structures:**

- Arrays

**Constraints:**

- 50 Kilobyte G-Code file input into program memory

# 8 Technical Problem Solving

## 8.1 Double Contour Problem

The method of detecting contours looks for the edges of the object to be detected and Maverick Bautista was the one who identified the issue. If the object is hollow, it results in parallel contours due to the edge detection. Meaning that there is a singular path, however there are two contours, in pink that are drawn which is shown in Figures 8.1.1 and 8.1.2.

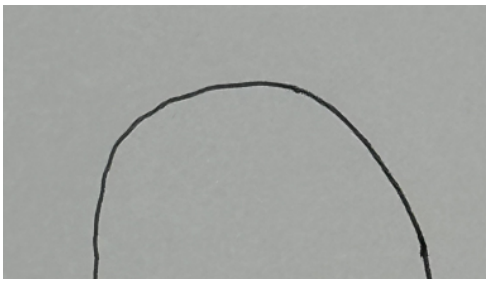

**Figure 8.1.1** Example Object

**Figure 8.1.2** Parallel contours

## 8.2 Double Contour Solution

Maverick Bautista implemented a skeletonization algorithm to change the original image by using morphology and erosion methods from online resources. The average distance between points is taken to create a new path coordinate. There is still an issue for images that are filled in that will be discussed in 8.3 and 8.4.

## 8.3 Bolded/Filled in Images Problem

With further testing, Maverick Bautista discovered a new issue regarding shapes or irregular forms of drawings that are filled in. It is apparent when testing filled shapes in Figure 8.3.1, and the result afterward is seen in Figure 8.3.2. Sketches that are bolded or filled in causes the generated skeletonized paths to be incorrect.

**Figure 8.3.1** Shape example

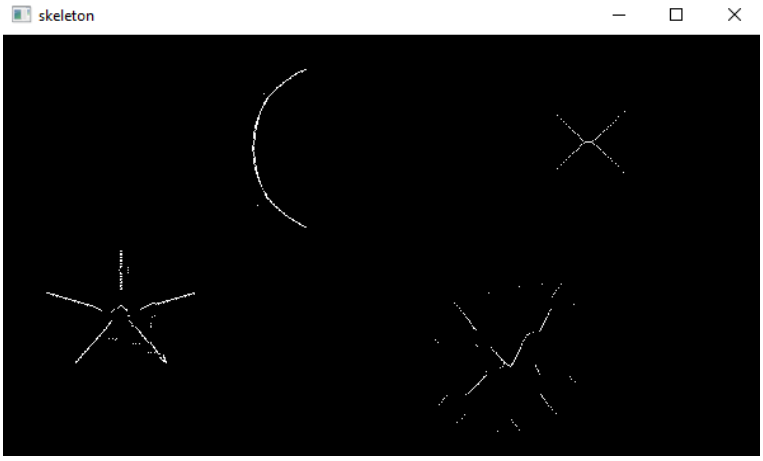


**Figure 8.3.2** Skeleton result

## 8.4 Bolded/Filled in Images Solution

The issue occurs due to the average between parallelized contours that generates the new path. Maverick Bautista did not construct a solid solution, but a temporary workaround was implemented for edge-detected contours as an alternative. The substitute process is only doing the edge cases. To refrain from this, it is suggested that users draw simple handwritten sketches and avoid filled-in shapes. Bolded cases work, but due to the skeletonization, some bolded images will leave curved contours on the edges which is seen in Figure 8.4.1 and Figure 8.4.2.


**Figure 8.4.1** Bolded text example


**Figure 8.4.2** Skeletonized bolded text example

## 8.5 Keeping Track of Paths Problem

The method used to obtain all the paths in the SVG file and put them into a list originally was not an issue until the need to convert coordinates came into play. Edward Haro found that while originally each new path is marked by an ‘M’ at the beginning, that M has to be removed to convert values, and with that M removed there is no way to distinguish between paths.

While the converted values are correct, without the M demarcation seen clearly in the first list in Figure 8.5.1, with a closer view in Figure 8.5.2, the paths are indistinguishable in the new list in “Printing Converted Values Test” in Figure 8.5.1.

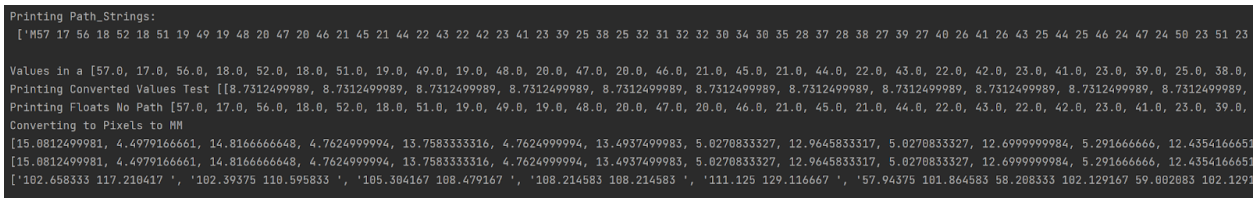


**Figure 8.5.1** Generated Converted Coordinate Values

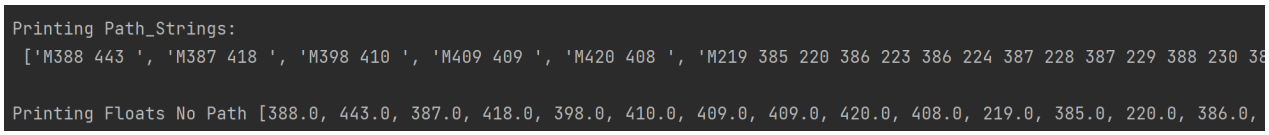


**Figure 8.5.2** Original Output of Paths and Converted Values

## 8.6 Keeping Track of the Paths Solution

Edward Haro found that the solution to this problem comes at the beginning of reading through the SVG file, with the first iteration essentially only creating one large list of the paths. The solution was to first, remove the ‘M’ from each new path, then split it to essentially put it in a main list, then put that into the larger list, and so on until the main list was full of all the paths. From there it was possible to iterate through the nested lists, convert the values to floats, convert them to millimeters, then add them back into the nested list one by one, and repeat for every nested list within the main list. Most importantly, this allowed for the path to be distinguishable without having the “M” inhibiting conversion.

## 8.7 Connections Problem

Antonio Garcia was responsible for reverse engineering connections on the prebuilt PCB. The PCB is based on the open source EBB board where the modified version has no schematic. Led to unknown connections making me unable to move any motor since the connections were unknown at the time. Procedure taken was to test various pins to check which power the output pins connected to the motors, but the motors continue to have power as long as the enable pin is on which ended the first attempted solution a failure. Next method of tackling the problem was to step an entire port one at a time which also failed, and lastly went through testing each pin at a time individually until there was some sort of output.

## 8.8 Connections Solution

Antonio used a multimeter and reprogrammed the MCU to turn pins on and all the other pins off to establish continuity connection. This allowed for pins to be found which made the stepper motors function. Continued testing on 3 pins with connections to the stepper motors resulted in findings leading to the stepper motor enable pins and the PWM pin for the servo motor [15].

## 8.9 The Shadow Problem

Maverick Bautista identified an issue with image processing to determine what objects were in the image to be considered a contour. It was prevalent for shadows, resulting in an adequately well-lit environment. A lamp brightens the scenery and decreases the chance for shadows to be detected.

## 8.10 The Shadow Solution

Maverick Bautista followed a suggestion from a fellow student to use adaptive thresholding instead of Otsu thresholding that classified the objects detected into two classes. Figure 8.10.1 shows the code for adaptive thresholding. After implementing adaptive thresholding, the process now ignored shadows, thus eliminating the need of a lamp.

```
binary = cv2.adaptiveThreshold(bilateral, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,5)
```

**Figure 8.10.1** Adaptive thresholding code

However, there is an issue when running this method, the region being determined has changed and has caused the skeletonization process to conduct edge detection which needs to be fixed.

## 8.11 The Conversion Problem

This issue is a smaller scale issue which was resolved and seen in the previous 8.5 and 8.6. In greater detail, Edward Haro found that the values are not converted to millimeters without first having the coordinates as floats, and that could not be done with the ‘M’ at the beginning of each path.

## 8.12 The Conversion Solution

Edward Haro found that the solution to this was to remove the ‘M’ at the beginning of the path line in the SVG file, then split the line and add it into the main list. Removing the ‘M’ got rid of the error that prevented the coordinates conversion to a float integer.

## 8.13 The Z Axis Problem

Edward Haro found that there was an issue with the Z-Axis coordinates when converting the SVG to G-Code. At first the code seemed to work fine when tested with the singular test file.

Once further testing was done with other SVG files, it was visualized that the simulated output had different paths drawn on separate planes as opposed to the same plane like it was supposed to. An example of this is the first test Q converted as seen in Figure 18.13.1:

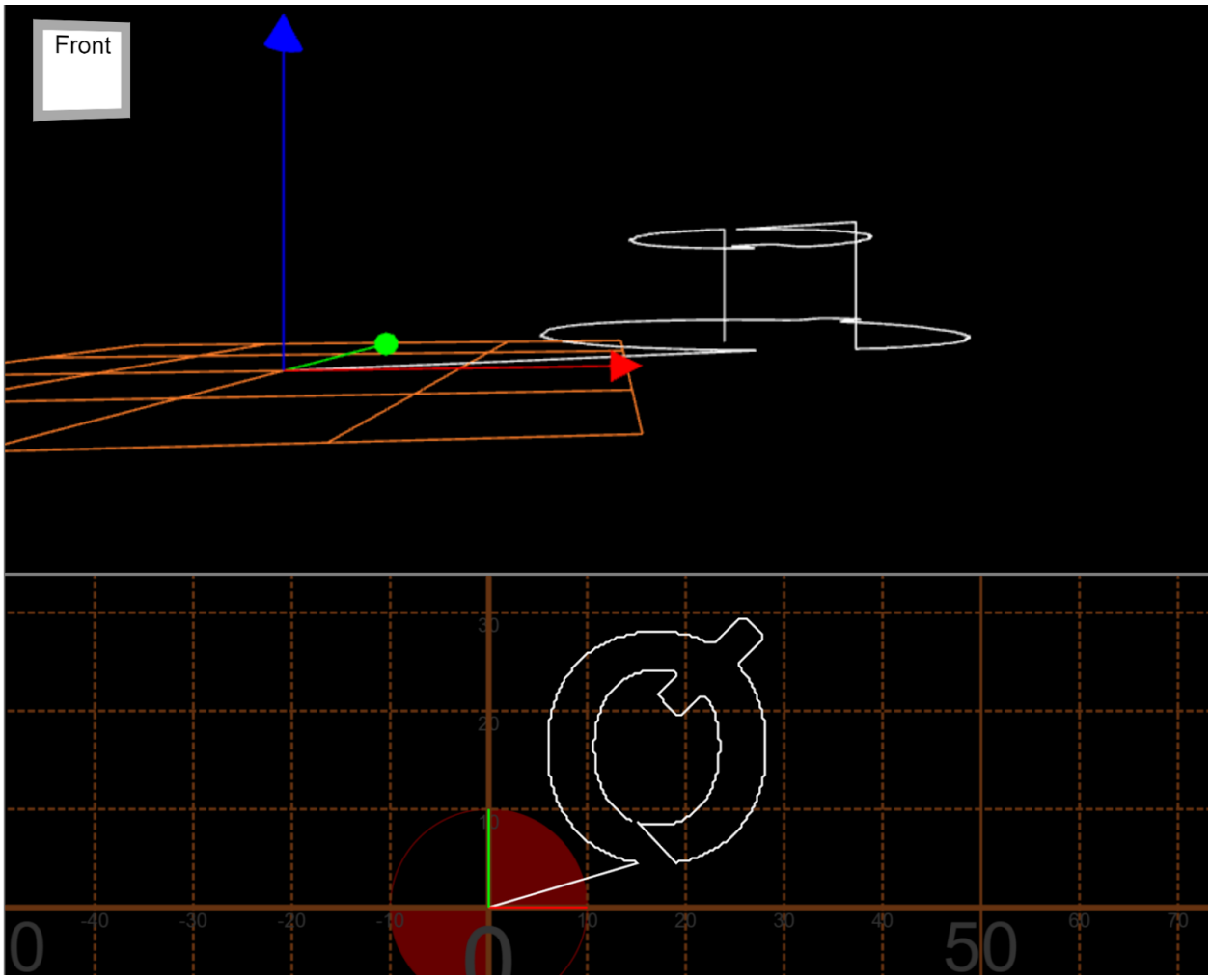


**Figure 18.13.1** Simulated G-Code

When trying to fix these issues, other problems were encountered with continuous “Z down” commands leading to paths continuing to spiral downward in the simulation. The issue lay in how the implementation alternated going up and down.

## 8.14 The Z Axis Solution

Edward Haro found that the solution to this problem lay in how the code decided when to lift or lower the pen. Initially it was supposed to be at every path, and while this was the case there were circumstances not thought about beforehand. The Z coordinate was meant to go down after the end of the first path, then up, then back down. In reality the pen had to:

- Move to the first coordinate in the path
- Go down
- Proceed to the end of the path
- Lift up
- Proceed to the first coordinate of the new path
- Go down, repeat

While the code only had the z axis change for beginnings and ends of new paths. Issues with spiraling came down to bugs in the code itself and how it was being modified during the trial and error process. The main issue came down to a misunderstanding of how to traverse the attributes within the nested lists. The code was fixed and modified by Edward Haro to follow the order of events listed above.

## 8.15 Stepper Motors Unnecessary Noise Problem

Antonio Garcia found that the stepper motors were moving slowly and created a large amount of noise. The issue was labeled as unnecessary to worry about until larger portions of the project were done as this was found early on. Multiple setups were tested with the motor controls, but the most effective at reducing the noise was decreasing the delay.

## 8.16 Stepper Motors Unnecessary Noise Solution

The motors continued to make noise at a drastically decreased amount. When members were attempting to establish communication between the PIC and a laptop using UART, when there was a frequency error on the software side of the microcontroller. Antonio changed the code into the PIC18F from 4 MegaHertz to 8 MegaHertz which caused the noise situation to return. The solution found was dividing the 2 milliseconds by a factor of 8 finalizing the step speed to .25ms.

## 8.17 Lack of Feedback Problem

Antonio Garcia handled the lack of feedback in the system as it made testing very difficult to debug problems in the code which is meant for the motors to move the pen. Functions were written and tested to verify if they work as expected in a loop, but do not work as intended.

## 8.18 Lack of Feedback Solution

The solution found was to include snippets of code which were tested with successful results. The one mostly used was the function which moved the servo up or down to indicate which path was being taken in a larger condition based loop and why it was not functioning as intended.

## 8.19 G-Code Integration Problem

Antonio Garcia handled G-Code input into the PIC18F MCU. Having the MCU properly decipher G-Code and read through the array was a simple task of filtering unnecessary

characters, until the problem was reached of reading in each character at a time. That led to a couple of problems, most notably were properly shifting each character to its appropriate position for its position, reading in a decimal, and when to pick the pen up and down.

## 8.20 G-Code Integration Solution

Reading in G-Code required filters in order to go through input until certain conditions were met and then the MCU goes through conditional functions to determine what command to process, be it XYZ, then for XY to know how much to left shift the first integer by which is until the decimal gets read in. Once the decimal gets read in the integers start being right shifted appropriately until the end of the integer then reset the used checking variables. All integers being placed into a position holding variable were subtracted from the ASCII character ‘0’ in order to change it from its char version of the integer to the appropriate integer version.

## 8.21 Servo Motor Problem

Antonio Garcia was responsible for servo motor control and found a problem with its operation. As mentioned previously, part of the previous solution was to create checks throughout the function in order to filter which commands the MCU should skip over, but the decimal check was not functioning as intended. The decimal check was not working as intended after the first decimal point was read. In the first iteration of using the motors to print out a triangle using G-Code the servo motor moved numerous times, but the input intended for the servo motor to move only 5 times. The problem was first discovered while Antonio was speaking to fellow team member Edward Haro while working on the project when during a test run Haro pointed out the servo motor being called upon a significant amount of times for the G-Code input.

## 8.22 Servo Motor Solution

The problem was lying inside the servo motor conditional function which had no decimal check reset, so the solution was to reset the decimal check variable whenever a space or a semicolon was read from the G-Code array.

## 8.23 Sending File to Arduino Problem

Edward Haro found an issue with the file not being sent, and no input was received from the Arduino. This issue was ongoing despite various attempts to correct it.

## 8.24 Sending File to Arduino Solution

Edward Haro implemented a solution by sending the file character by character using a loop, and changing the encoding to ASCII. This way, the file transmission took a long time, but the file was still successfully transferred. In theory, the transmission could stop to give the microcontroller time to be able to read in more information later if the file was too big.

# 9 User Interface Design

## 9.1 Application Control

Maverick Bautista implemented a user interface within the Python script. The process begins with an IDE utilizing OpenCV. Once the user runs the IDE, a GUI from the Tkinter library is used in conjunction with generic windows from OpenCV. All Tkinter windows must be followed correctly or the program will run into a runtime error. The OpenCV windows can be closed by pressing any button on the PC system. A button window is prompted before the File Explorer program will open for the user to select a file. After selecting a file, the image will be converted to a gray scale, followed by binary, and inverted binary conversion to which the user will follow the process with pop-up windows. The image will then be parsed to be converted to contours and a new window highlighting the contour lines in pink and with bounding rectangles will appear. Finally Tkinter windows stating the amount of paths and detected contours emerges.

Antonio Garcia inputs the G-Code into MPLAB X IDE, specifically into the const char G-Code array. The array places the information into the program memory instead of the SRAM which allows for greater storage capacity.

## 9.2 User Interface Screens

When running the Python script, the first interface is a Tkinter popup window seen in Figure 9.2.1, this window is a button that requires the user to select a raster image. After clicking the button, the user is taken to the file explorer window seen in 9.2.2. Once the user selects an image, the file is converted to greyscale that is displayed in Figure 9.2.3. Figure 9.2.4 then displays a binary image and then an inverted binary in Figure 9.2.5. Afterwards, a bounding box to show the identified object is shown in Figures 9.2.6 and 9.2.7 applies the detected contour path in pink. A Tkinter window then appears with Figure 9.2.8 stating the number of coordinates listed. Finally, the last interface is a Tkinter window seen in Figure 9.2.9 stating the total number of continuous contours detected.

For Figure 9.2.1, the user must click on the button that states “Please select an image/file, click to close.” to continue the process, if the user clicks the close button, the window will reappear until an image is selected. A similar process occurs with the file explorer window that will continuously appear if the close or cancel button is clicked. From Figures 9.2.3 to 9.2.7, the windows proceed when the user provides any input such as a keystroke or mouse buttons. Figures 9.2.8 and 9.2.9 are Tkinter windows which require the user to click the close buttons to exit the window.

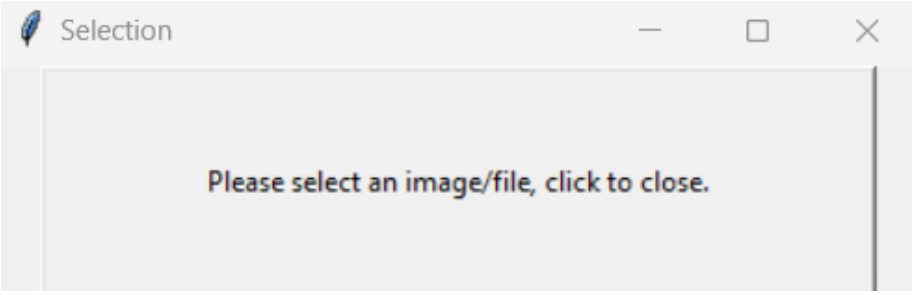


**Figure 9.2.1** Window popup to instruct image selection.

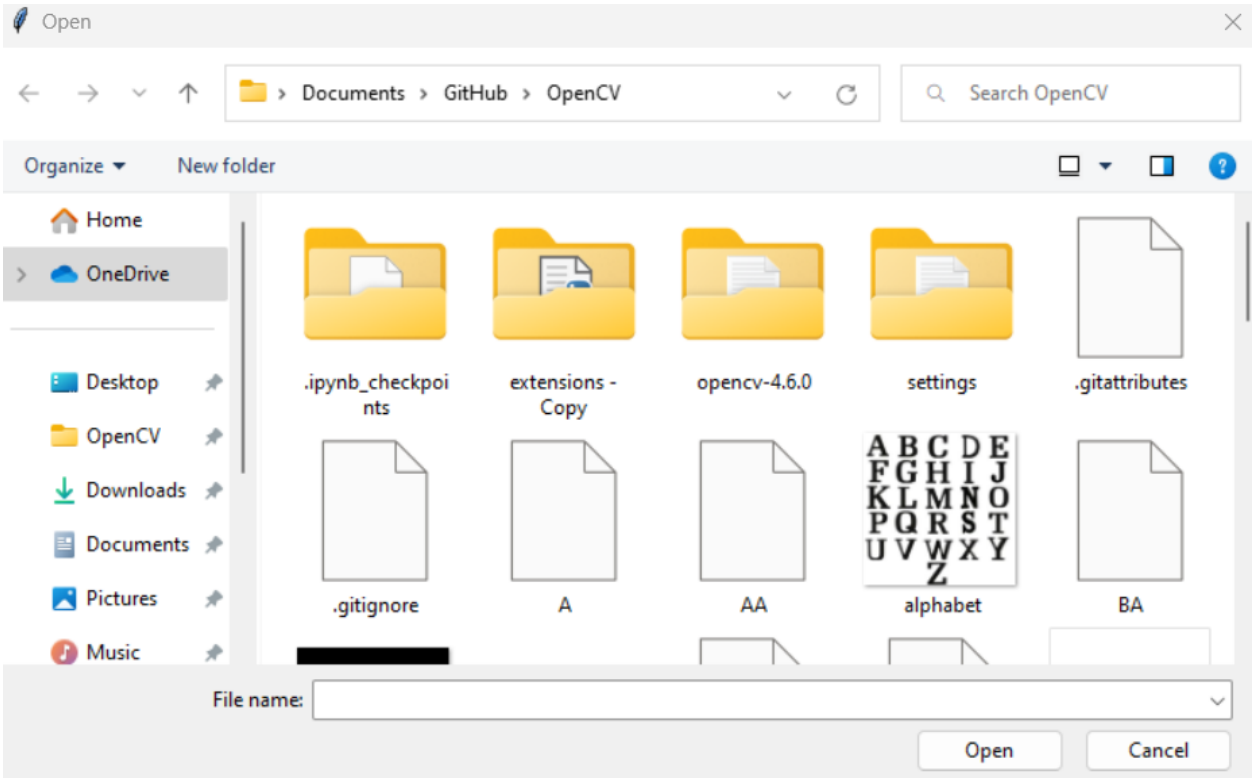


**Figure 9.2.2** File Explorer popup for file (image) selection


**Figure 9.2.3** Sample image converted to grayscale.


**Figure 9.2.4** Sample image converted to binary.

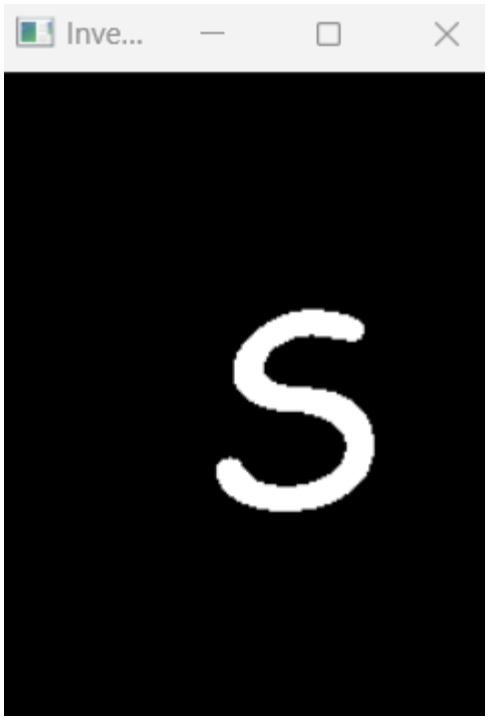


**Figure 9.2.5** Sample inverted binary image.


**Figure 9.2.6** Restored image with bounding rectangles.

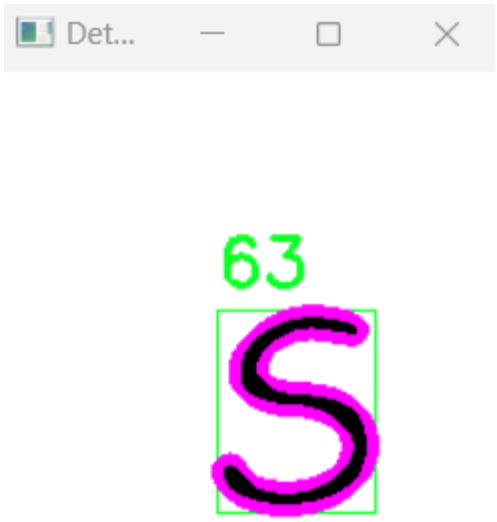


**Figure 9.2.7** Detected contours colored in pink with bounding rectangles.

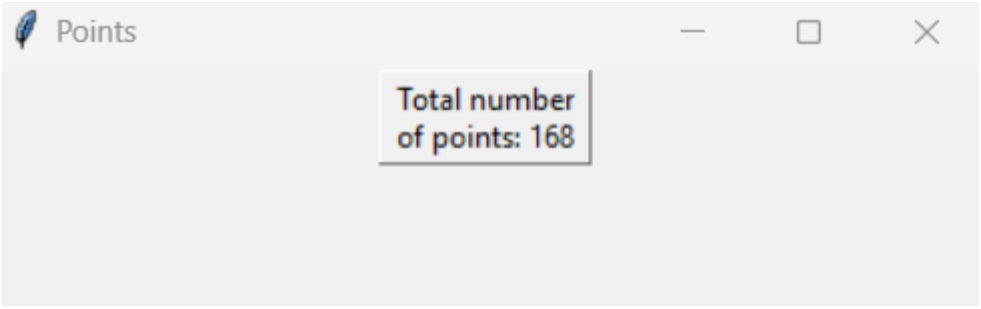


**Figure 9.2.8** Total number of coordinates listed.

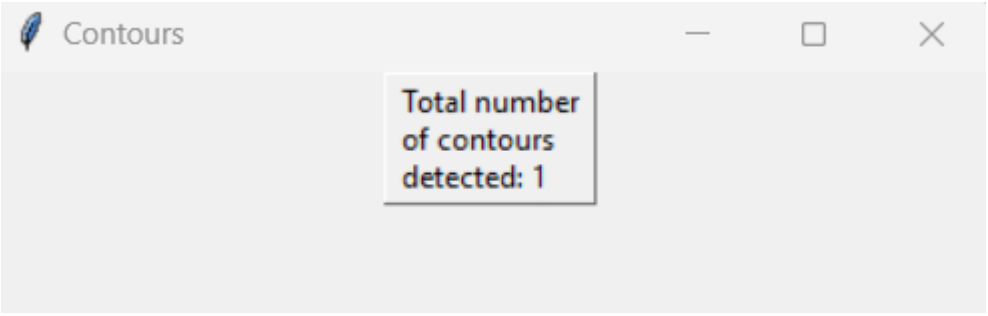


**Figure 9.2.9** Total number of contours listed.

# 10 Test Plan

## 10.1 Test Design

**Test Case 1**

1. The objective of the experiment is to see if the coordinate paths of the detected contours will be placed in an SVG file. If functional, this test validates that the raster image is taken and will be an SVG output.
2. The setup will be on the Jupyter Notebook IDE that runs the OpenCV library.
3. The IDE runs to conduct image processing to convert from raster to SVG where multiple GUI windows highlight the conversion. Once it runs, the path coordinates are listed in an SVG file that can be accessed. To visually test and verify it, an external software that can read and display SVG files such as Inkscape is used.
4. The expected result should be the SVG file accessed by Inkscape and can visually show the converted SVG with the pink lines indicating the paths that should be drawn.

Maverick Bautista is responsible for test case 1 with the setup of the experiment on a PC system to run the Python script on Jupyter Notebook. To verify that the experiment is successful, Inkscape was used to open the SVG file and see what appears [16].

**Test Case 2**

1. The objective of the experiment is to see if the SVG file can be successfully converted into working and understandable G-Code. If the output file can be simulated and gives the correct results, then the test validates that the G-Code conversion is successful.
2. The test setup is on the PyCharm IDE, running the xml library that comes with Python already.
3. The PyCharm IDE runs to open the file, parse through it, get the number of paths and output them, and then modify them enough to be ready for conversion. Once modified and put in nested lists, the path coordinates are converted to millimeters. From here a new file is written, appended necessary G-Code commands as well as X and Y to the corresponding coordinates, and the file is closed once it is done being written. To visually verify the file and test it, it is opened on a G-Code visualizer/simulator and inspected to see if the path created matches the SVG image, and if it is all on the same plane [17].
4. The expected result is that the SVG file is correctly and accurately converted into a G-Code file that accurately recreates the SVG image and can be read by G-Code toolpath simulators.

Edward Haro was responsible for testing the SVG to G-Code conversion Python script to produce accurate G-Code files.

**Test Case 3**

1. The objective of this test is to see if the SVG process and G-Code process can be integrated into one Python script. Once done, it allows the process to be done in one PC system and to be done more efficiently in one script.
2. The setup will be on the Jupyter Notebook IDE that runs the OpenCV library.
3. The IDE runs to conduct image processing for conversion to an SVG file. It is then parsed for G-Code commands to be written and a new file is created. To verify the process, an online simulator or other software can be used.
4. The expected result is that both an SVG file and a G-Code file will be created and output.

Maverick Bautista and Edward Haro were responsible for testing the Python script to produce SVG and G-Code files.

**Test Case 4**

1. The objective of this test was to see if the motors were capable of moving properly with mm input instead of steps for input.
2. The setup will be done on the MPLAB X XC8 console with the help of the PICKIT4 to reprogram the MCU.
3. The PICKIT will send the configuration and code for the MCU and process the hard set distance the motors should move.
4. The expected result is that both the x motor and y motor move the specified amount of mm in their respective direction.

Antonio Garcia is responsible for the motor control system and the motor commands including the software and hardware.

**Test Case 5**

1. The objective of this test is to see if the G-Code can be outputted and read by the microcontroller. Methods of input vary between Serial and inputting files manually.
2. The setup will be on both the PyCharm IDE and the PICKIT4 using the MPLAB X XC8 console.
3. Ideally the PyCharm script will send the file character by character to the MPLAB X, where it will put each character into an array and then read through it later. Otherwise, the file will be manually inputted into the console and then the microcontroller will be reflashed with the new commands. Its success is verified by visually inspecting what the machine outputs and draws.
4. The expected result is that the machine will output and draw the G-Code commands given.

Edward Haro and Antonio Garcia worked together on test case 5 for G-Code input into the microcontroller with serial communication being a failure and manual input resulting in a success.

**Test Case 6**

1. The objective is to run all modules in a swift process to convert an image to SVG, SVG to G-Code, and the plotter to read in the G-Code and begin moving.
2. The setup will involve a PC system to run the IDE with OpenCV to run the Python scripts and to have the MPLAB X XC8 console to input the G-Code.
3. The IDE runs to convert to SVG, then to G-Code, finally it is inputted into MPLAB to read in.
4. The expected result is that the image is processed with G-Code being inputted into the MCU with the plotter physically drawing the image.

Maverick Bautista, Edward Haro, and Antonio Garcia worked together to test all functions on a single PC system.

## 10.2 Bug Tracking

Each member personally tracked any bugs or issues they found depending on the PC they were using. To track and save any code that was last worked on, the files were saved onto GitHub repositories. It also allowed the other members to clone and verify the code. If any important issues were found, it was discussed amongst the members on whose responsibility it

belonged to. Anything related to image processing was assigned to Maverick Bautista. Edward Haro worked on any issues on G-Code commands and communication. Antonio Garcia was tasked with hardware problems and programming the MCU.

## 10.3 Quality Control

Most of the recent testing was done near the end of the project as implementation between other modules was key to have a working prototype. Any issues that arose were due to software bugs or edge cases that needed to be addressed. Hardware was least of the concern as the iDraw 1.0 pen plotter came with everything required. If a member found a bug, they followed up with a solution to fix it. After determining a solution and implementing it, a new test is needed to see if errors are reproduced. If the bug was not present, it was determined to be resolved.

## 10.4 Identification of critical components

The PIC18F microcontroller, located on the EBB board is a critical component as its pins are connected to many different ports. Testing and verifying what ports the pins belonged to ensure the correct inputs and outputs were being applied when programming it. Other critical components included the 2 NEMA 17 stepper motors and a servo motor for movement. When testing, the torque and phase of the motors were considered as a high value causing loud noises when moving. However, with a low value, the motors hardly move. Finally the PICkit 4.0 enabled reflashing for the PIC18F microcontroller.

## 10.5 Items Not Tested by the Experiments

A breadboard was used to temporarily house resistors to safely test pin connections from the EBB to PC system but is not necessary following Test 6 as it is only used for the final build.

# 11 Test Report

## 11.1 Test 1

### 11.1.1

Maverick Bautista is responsible for the experiment of running the Python script to test different varying SVG files.

1. After running the IDE, a new file called “path.svg” is generated in Figure 11.1.1. Whenever the IDE runs again and converts an image, the following file will be rewritten based on the newly generated paths for it.
2. The SVG file is defaulted to open the Microsoft Edge Browser which shows in Figure 11.1.2. However due to the size of the original image when compared to A4 size, there is a lack of space to actually display it. To get a proper visualization, software that can read SVG files such as Inkscape is needed which is shown in Figure 11.1.3 to verify the result.
3. After selecting the SVG file in Inkscape, the image is scaled to A4 size. The issue when opening Microsoft Edge with a blank white space is due to the size of the original image. The distinct shapes are identified and based on the edge detected contours which are seen in the pink paths.
4. Although this is great to see the converted paths, the ideal method required the skeletonization process.


**Figure 11.1.1** Generate SVG file

**Figure 11.1.2** SVG file when opened on a browser

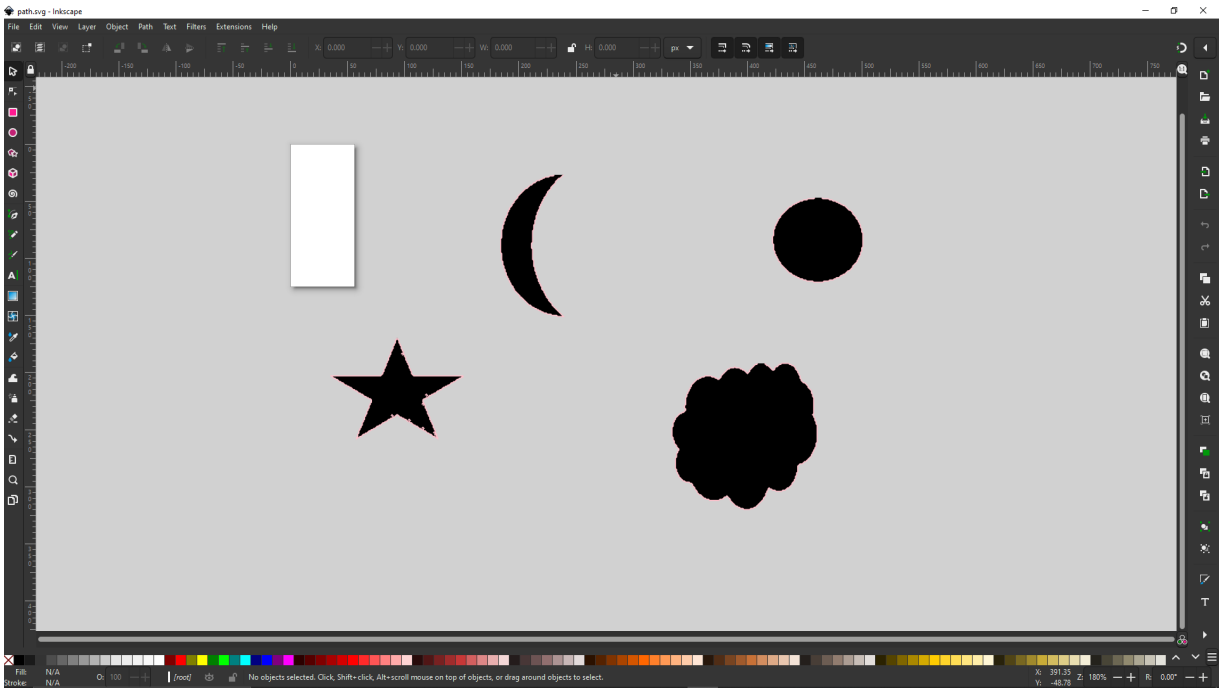

**Figure 11.1.3** Shape example seen in Section 8.3

## 11.1.2

Maverick Bautista is responsible for the new test to implement a skeletonization method for contour detection.

1. The process is the same as 11.1.1 with modifications to the Python code to conduct skeletonization.
2. After running the Python script, the newly processed image should now be skeletonized, meaning the image has reduced pixels in the foreground of the path.
3. With the script executed, a test comparison of Figure 11.1.4 and 11.1.5 shows the skeletonization reduction. There are issues with the pathing that is due to the average of coordinates of the original image and the new path generated by that skeletonization process. It is prevalent on images that are bolded or have a filled in background.
4. The current iteration seen is the one used primarily for text and sketch detection to determine the contours and convert it to an SVG file.

**Figure 11.1.4** Intersecting and Bolded letter example of a Q

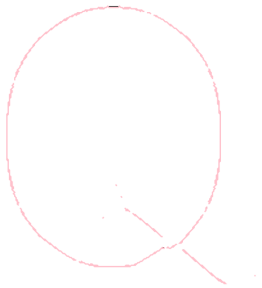

**Figure 11.1.5** Generated skeletonized Q

## 11.2 Test 2

Edward Haro is responsible for experimenting with the Python script to convert SVG files to working G-Code files.

1. After running the PyCharm IDE, a new file called “svg.gcode” is created. Each time the IDE is run the file is overrun and changed with the new contents. The new contents being the new G-Code commands and coordinates.
2. The toolpath generated in Figure 11.2.1 matches very similarly to the SVG image received from Maverick. Ideally, the only real difference between the files is the size is converted from pixels to millimeters.
3. The results match very closely if not exactly to what was scanned and received from the SVG file, even with the added lines and dots that the original image to SVG conversion picked up.
4. In this test case, there are no corrective actions needed that were not already addressed in Section 8, as in this example all issues with conversion were already fixed. The only additional correction includes changing the file to a single line to be able to be pasted into MPLAB X, as well as changing the ups and down values for Z to positive 2 and 3.

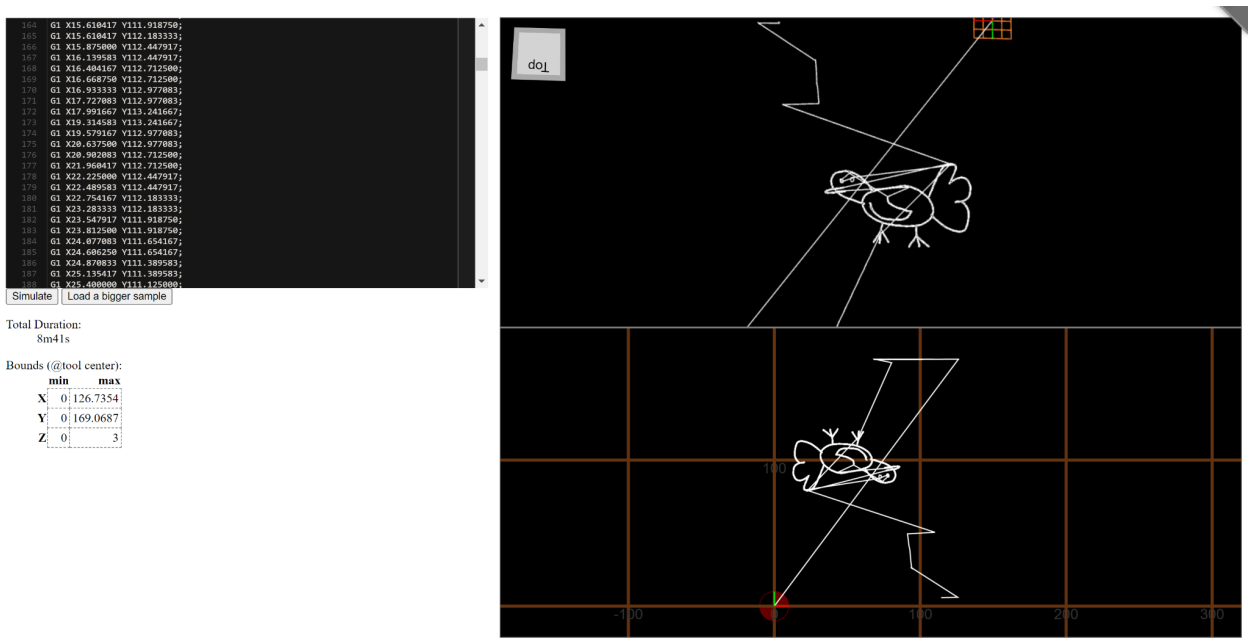


**Figure 11.2.1** Simulated G-Code on an online resource

## 11.3 Test 3

Maverick Bautista and Edward Haro are responsible for running the conversion process in a single Python script.

1. Edwards G-Code script is integrated into Maverick's IDE to run the conversion process.
2. It is expected that the motors would follow the path movement and include the servo motor moving up and down to represent the utensil.
3. The G-Code file seen in Figure 11.3.1 is selected and Figure 11.3.2 displays the G-commands. With the generated file, CAMotics and a GitHub G-Code generator are used to check a 3D environment to visualize the path.
4. No corrective actions were taken.


**Figure 11.3.1** G-Code file that is created and written into

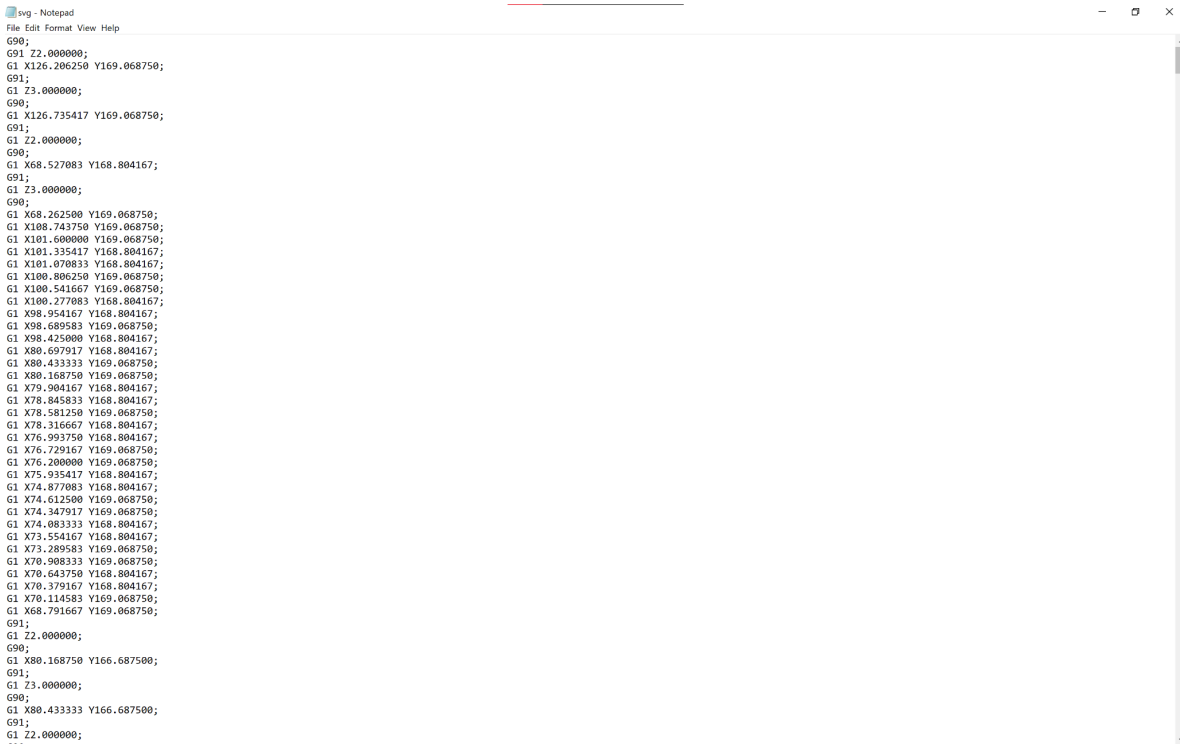


**Figure 11.3.2** G-Code commands file

## 11.4 Test 4

**Antonio Garcia**

1. The test performed as expected and was successful as the motors moved appropriately moving 30 millimeters along the x-axis, and 15 along the y-axis.
2. The test results matched the input and were measured with a ruler on hand.
3. Motor commands work as intended and are able to convert mm into steps for the motors to move accurately.
4. No corrections were made for this test.

## 11.5 Test 5

Antonio Garcia and Edward Haro are responsible for testing the execution of the G-Code commands by the motors.

1. The test is unsuccessful as the output is supposed to be a single triangle. The result was part of a triangle with additional lines that were not part of the G-Code file as seen in Figure 11.5.1.
2. Expected results were a simple triangle with complete connections and the results did not resemble the expected result enough.
3. The test shows errors within the motor command portion of the code as the servo motor moved a tremendous amount instead of the designated amount provided from the G-Code.
4. The corrective actions taken by Garcia was to change the z-axis code which specifically worked with the servo motor.

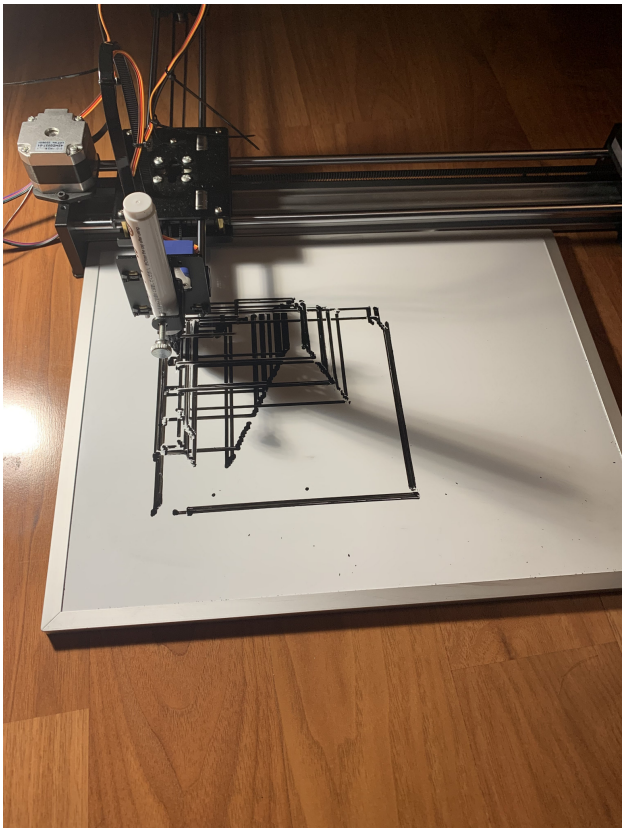

**Figure 11.5.1** Motor movement test result

## 11.6 Test 6

Maverick Bautista, Edward Haro, and Antonio Garcia are responsible for the experiment to test everything in one successive process.

1. After proceeding with the test plan, the plotter begins to move to draw the sample image of a bird.
2. Since motor movements are confirmed to properly read in G-Code, the output example of the bird is seen with it being drawn in a few minutes in reference to the video linked in the Appendix G.
3. It is seen that the drawn image that the plotter sketches resembles the original doodle created in the right of Figure 11.6.1.
4. No corrective actions were taken.

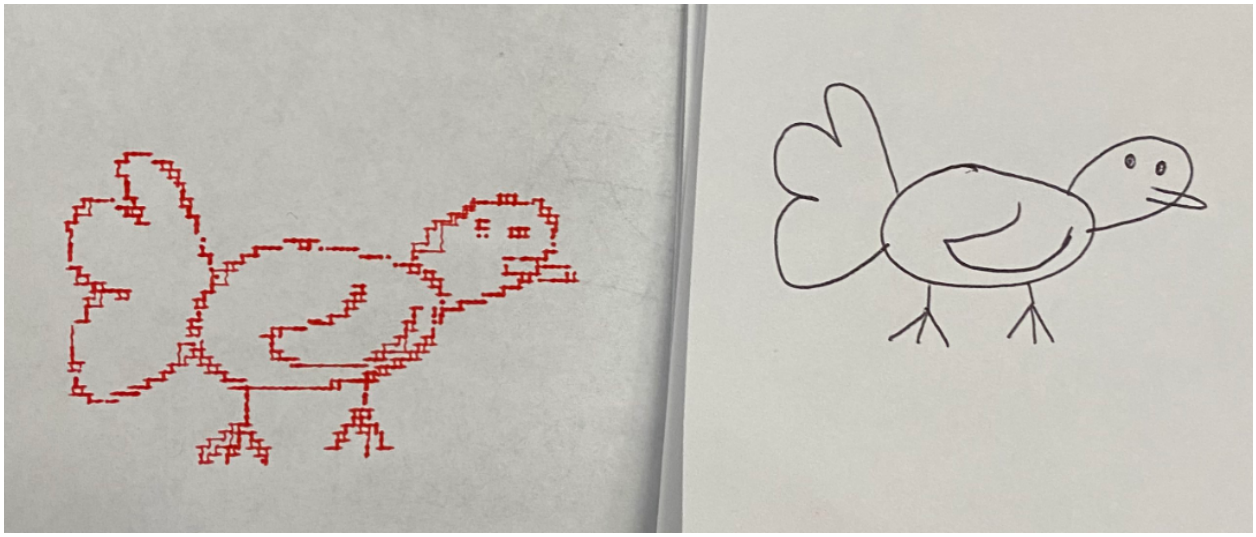

**Figure 11.6.1** Sample hand drawn image to plotted sketch

# 12 Conclusion and Future Work

## 12.1 Conclusion

By working together as a team and communicating with each other, we were able to construct a working prototype within the two quarter time frame. All the members often saw each other and were in frequent contact to discuss any issues or concerns. The weekly lab sessions also gave an opportunity to get feedback from our section advisor and see what we can improve upon. We were able to get computer vision to work at a fair margin with the G-Code process able to parse through an SVG file without external libraries, and to have the motors follow the commands. Each member had a difficult and lengthy process of bugs and issues that needed to be resolved in time. Thankfully it worked out in the end with a few complications that can be fixed in future work.

**Maverick Bautista**

The beginning was a process of trial and error to get image processing completed. It was a slight challenge to learn Python but made the overall task to use computer vision much simpler. In terms of my technical objectives, I believe they were adequate given the task of recognizing raster images and converting them to SVG. Due to the method I used, it used edge detection which did cause some issues about parallel contours and the solution to the shadows caused problems for skeletonization. Regardless, the overall procedure to generate SVG was handled well to give to Edward and have a result. It was important to work as a team and communicate with the other members as it allowed feedback and suggestions from them. The project taught me a lot about engineering applications and how it is used for those in the field of electrical engineering.

**Edward Haro**

The G-Code conversion process was successful after many attempts and through much trial and error. Learning Python and brushing off my coding abilities were taxing as problem solving and coding have never been my strong points. When testing and debugging, many iterations of code and ideas are run through, with new problems arising each time. The method I used for G-Code conversion is very specific to our use case, with the original output being able to run through simulators, but the final output being specifically designed for MPLAB X to read through as easily as possible. Due to this being an XY plotter, there are no f or s command and no spool or spindle speeds, only G Commands. The output worked well with Antonio’s part and the code was successfully read. Despite the serial communication not working with the MCU in the end, the file was able to be sent to an Arduino character by character, indicating a success in transferring. Handshaking was never successfully tested with the Arduino either, but in the end we were able to establish a working product that only lacked communications. Teamwork was more important than ever on this project as we all needed to successfully communicate exactly how our parts interact with each other and what we needed to accomplish. Feedback was important and everyone had something to contribute to the other's work, even if it was just suggestions or comments. The project taught me a lot about how to apply what I have learned at the school and how to apply those to the engineering field to successfully complete a task and project.

**Antonio Garcia**

The team based project was a very successful method to introduce working on a semi-long term project, around 6 months, with a small group. The project created from the group met the overall project goals which were established. There was a technical design which failed to be implemented which is the UART communication with the MCU and computer as there was an inability to synchronize the frequencies accordingly with the baud rate which was a failure on my part. The overall project enabled me to learn how to read a microcontroller data sheet and work with an entirely new IDE environment, MPLAB X, and console, XC8. During the creation of the project we had to communicate with each other on a weekly basis to give each other an idea of how far we have gone in our portion of the project. Maverick did a great job working with Python, OpenCV, and SVGs for the first time. Edward was successfully able to then take the SVG input and output as a G-Code file which is reprogrammed into the MCU using the PICKIT4 and move the motors as intended.

## 12.2 Future Work

With the wide array of examples and online resources available, there are many things we wish to incorporate beyond the limitations of 2 quarters. We would like to reintroduce the possibility of incorporating a utensil selector that the user can choose. Alongside this ability, it allows us to work on further enhancements for color detection that includes on the fly switches for the utensil. Meaning we can halt the process during a sketch to change the type or color of utensil.

It was a tremendous task to learn Python and also work on image processing to utilize OpenCV. Contour detection was a critical aspect to edge detection, I believe that future work from either us or future students can improve upon it. Just like CNC machines there are methods to fill in sketches, specifically for bolded text, that we hope to incorporate in our future rendition.

Creating a less pixelated output is a part of the future of the project and is under implementation. It is implemented by reading through the G-Code array then creating a curve by moving both motors at different paces in order to create various curves by creating a mathematical equation which goes through all points. This would not occur if points were on the same line.

An issue that we would like to resolve is the communication to enable UART. The process would allow seamless transition after converting an image. The MCU would continuously monitor data from the serial port to be read in. Our current process involves manually inputting the G-Code commands into a temporary array.

Overall, these are but a few additions we would like to include for the Handwriting Copying Machine. Plotters have a long history of development and hope that this inspires others to take interest in it.

## 12.3 Acknowledgement

We would like to acknowledge and give credit to the aforementioned individuals or groups who have helped us in this project.

1. Cody Simons as the direct point of contact and teaching assistant who guided us through the two quarters. He has provided online resources and engineering methods to conduct our research and work.
2. Roman Chomko as the advisor and professor who suggested our contingency plan and

method of programming the PIC controller.

3. Manglai Zhou in charge of the Makerspace who provided us materials and access to soldering equipment.
4. iDraw Website for the general setup and access to the pen plotter.
5. Dejan - How to Mechatronics who provided the general layout and resources for the contemporary plan.

# 13 References

[1] "Ender series 3D printers," *crealitystore-eu*. [Online]. Available: https://store.creality.com/eu/collections/ender-series-3d-printer?gclid=CjwKCAjw_MqgBhAGEiwAnYOAer7cVn2FnE-NhWF5oo3KTneD-A_78jjUSP1G7QHyRP3yp45Jvc1jlxoCS4kQAvD_BwE.

[2] Evil-Mad-Robot, "AxiDraw v3," *Evil Mad Scientist Shop*. [Online]. Available: https://shop.evilmadscientist.com/productsmenu/846.

[3] Eco home office, "Robot House A3 Working Area metal drawing robot kit writer XY plotter idraw hand writing robot kit based on 3D printer Corexy or HBOT structure support laser engraving," *Eco home office*. [Online]. Available: https://ecohomeoffice.com/products/robot-house-a3-working-area-metal-drawing-robot-kit-writer-xyPlotter-idraw-hand-writing-robot-kit-based-on-3d-printer-corexy-or-hbot-structure-support-laser-engraving.

[4] "IDraw 1.0 pen plotter/handwriting drawing machine/XY plotter," *UUNATEK*, 02-Dec-2022. [Online]. Available: https://uunatek.com/product/idraw-handwriting-drawing-machine/.

[5] "Reprap machines," *RepRap*. [Online]. Available: https://reprap.org/wiki/RepRap_Machines.

[6] "Microchip technology." [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/39931d.pdf.

[6] JOHN, "G-code and M-code command list for CNC Mills [ examples & tutorials ]," *CNCCookbook*, 19-Aug-2022. [Online]. Available: https://www.cnccookbook.com/g-code-m-code-command-list-cnc-mills/.

[7] flopflop 3555 bronze badges, "UART and internal clock on pic18f4431," *Stack Overflow*, 01-Jul-1962. [Online]. Available: https://stackoverflow.com/questions/32602715/uart-and-internal-clock-on-pic18f4431.

[8] JOHN, "G-code and M-code command list for CNC Mills [ examples & tutorials ]," *CNCCookbook*, 19-Aug-2022. [Online]. Available: https://www.cnccookbook.com/g-code-m-code-command-list-cnc-mills/.

[9] "UART baud rate: How accurate does it need to be? - technical articles," *All About Circuits*. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/the-uart-baud-rate-clock-how-accurate-does-it-ne ed-to-be/.

[10] K. Magdy, Dj, Qasim, Microdigisoft, and CredX, "Servo motor control with pic microcontroller - SG90 MG90S – MPLAB XC8," *DeepBlue*, 16-Nov-2019. [Online]. Available: https://deepbluembedded.com/servo-motor-control-with-pic-microcontrollers-pwm-pt1/.

[11] LaxmanLaxman 1 and ospiderospider 8, "Convert contour paths to SVG paths," *Stack Overflow*, 01-Feb-1964. [Online]. Available: https://stackoverflow.com/questions/43108751/convert-contour-paths-to-svg-paths.

[12] CNCCookBook, "G-code and M-code command list for CNC Mills [ examples & tutorials ]," *CNCCookbook*, 19-Aug-2022. [Online]. Available: https://www.cnccookbook.com/g-code-m-code-command-list-cnc-mills/.

[13] Real Python, “Python GUI programming with Tkinter,” *Real Python*, 30-Jan-2023. [Online]. Available: https://realpython.com/python-gui-tkinter/.

[14] *Microchip classic*. [Online]. Available: https://forum.microchip.com/s/topic/a5C3l000000MU06EAG/t345382?comment=P-2515736.

[15] K. Magdy, S. Sebastian, M. Endis, J. D, P. Oo, Jannie, and A. Mir, “Pulse width modulation - PWM tutorial: Deepblue embedded tutorials,” *DeepBlue*, 24-Apr-2020. [Online]. Available: https://deepbluembedded.com/pwm-pulse-width-modulation-tutorial/.

[16] Andy, Dejan, C. Extreme, M. Madushantha, Loic, and J. Green, “DIY PEN Plotter with Automatic Tool Changer: CNC Drawing Machine,” *How To Mechatronics*, 26-Feb-2022. [Online]. Available: https://howtomechatronics.com/projects/diy-pen-plotter-with-automatic-tool-changer-cnc-drawing-machine/.

[17] “Code q'n'dirty toolpath simulator,” *g*. [Online]. Available: https://nraynaud.github.io/webgcode/.

# 14 Appendices

**Appendix A:** Related Documents and Materials for Standards

- IEEE 754 - https://peps.python.org/pep-0754/
- MPLAB - https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide
- PIC18F family datasheet - https://ww1.microchip.com/downloads/en/DeviceDoc/39931d.pdf
- UART - http://ww1.microchip.com/downloads/en/Appnotes/Getting-Started-with-UART-Using EUSART-on-PIC18-90003282A.pdf

**Appendix B:** Parts List

- 2 NEMA 17 Stepper Motors
- 1 Servo Motor
- iDraw 1.0 Pen Plotter
- Utensil - Pencil, Pen, Marker, etc.
- Camera - Phone
- Lamp
- 12V DC 1A charger

**Appendix C:** Code Repositories

- https://github.com/Mavbrick/OpenCV
- https://github.com/Antoniog0/SeniorDeisgnpickit4.X

**Appendix D:** Equipment List

- Digital Multimeter
- PC System - Laptop
- PICkit 4
- USB-A to micro-USB cable

**Appendix E:** Software List

- Jupyter Notebook
- MPLab X IDE
- PyCharm IDE
- Inkscape

**Appendix F:** Arduino Communication

```
void setup() {
  Serial.begin(9600); // set baud rate to match Python
   // wait for serial connection
}

void loop() {
  if (Serial.available() > 0) {
   if (Serial.available()){
    char character = Serial.read();

    Serial.print(character);
    }
  }

  }
```

**Figure 14.1** Arduino Code for serial communication

## **Appendix G:** Videos

- Motor Movement: https://www.youtube.com/shorts/IR5rzqiG7-0
- Motor Movement: https://www.youtube.com/watch?v=tu_IAGkAYHk