Robotic Caricature Artist

By

Dylan Huang

Peter Kuimelis

Soumithri Bala

Final Report for ECE 445, Senior Design, Spring 2018

TA: Xinrui Zhu

May 2018

Project No. 2

Abstract

This document outlines the design and construction of the Robotic Caricature Artist, with three primary areas of focus: hardware and mechanical design, firmware, and processing software. The final product is meant to be used as an entertainment device in public settings such as county fairs and amusement parks; it consists of an easel mounted with belts, motors, and circuitry whose objective is to mimic the art of a caricaturist.

Contents

1 Introduction

1.1 Objective

A caricature is a style of portrait that exaggerates certain features of the subject for comedic effect. Modern caricature artists typically work as street vendors or at social events, creating caricatures of patrons for a small fee. But caricature artists have a skill that takes at least a year to perfect and train, making them inaccessible and rare, especially if you do not live in a city [1]. Since we do not have many hours, we wish to create an automated system that replicates the caricature artist experience as closely as possible. Caricature artists can take a mental snapshot of the subject and draw a simple portrait in a few minutes. Thus, our system should be able to capture an image of a patron, apply effects to the image, and use a motorized system to draw the image onto a sheet of paper.

1.2 Background

Image processing is ubiquitous in entertainment – applications such as Snapchat allow for one to immediately apply filters to an image, which has proved to be tremendously popular. One feature nearly all apps of this nature share is instant gratification; in some cases, filters can be previewed before the image is captured. The entertainment value of certain art, however, is being able to experience the process of creation as much as the result. Listening to music at a concert, for example, is completely different from listening to a pre-recorded track. This motivated our decision to have our project replicate the excitement and anticipation of watching an artist work rather than just receiving the final product.

Although our system also uses a camera for image capture, the audience will not receive a preview of the applied transformations before the drawing is completed. Next, rather than use a printer to create the drawing, we want to use a v-plotter system mounted to an easel, as this approach has many advantages. First, the illustration process is bared to the user, as the paper is exposed during the process. Further, the toolpath of a v-plotter will more closely mimic that of a human, as the machine will draw each discrete stroke rather than sweep the page along an axis. Finally, the drawing is also done with a pen, which adds a natural variation that a desktop printer lacks.

While the hardware of this project is incredibly important, the software will be responsible for integrating all the individual components into a cohesive product. The image processing component of our project will be done entirely in software; there is prior research in computer-generated caricatures which we are looking to implement [2]. Additionally, lower-level software, such as firmware, will be essential for reliable communication between the computers and microcontrollers in our project.

1.3 High-Level Requirements

- Apply a caricature effect to an image and generate plotter instructions
- Plot geometric shapes and complex lines given a G-code script
- Must perform the entire process autonomously

2 Design

2.1 Block Diagram

Our block diagram requires three modules: a power supply, control module, and I/O module. The power supply's purpose is to drive voltage to the control module to the Powered USB Hub, and ATmega328 custom circuit. The power supply ensures that our entire system is reliably powered and can be driven from any wall outlet. The control module consists of a Computer, Powered USB Hub, and ATmega328 custom circuit that will be able to accept inputs from a phone camera, caricaturize and vectorize an image, and driver stepper motors that will carry out the vectorized instructions and draw an image. The peripheral module includes the phone camera and may be expanded to improve user experience. The mechanical module consists of all the components that are needed to produce the final drawing: two stepper motors and a servo motor.

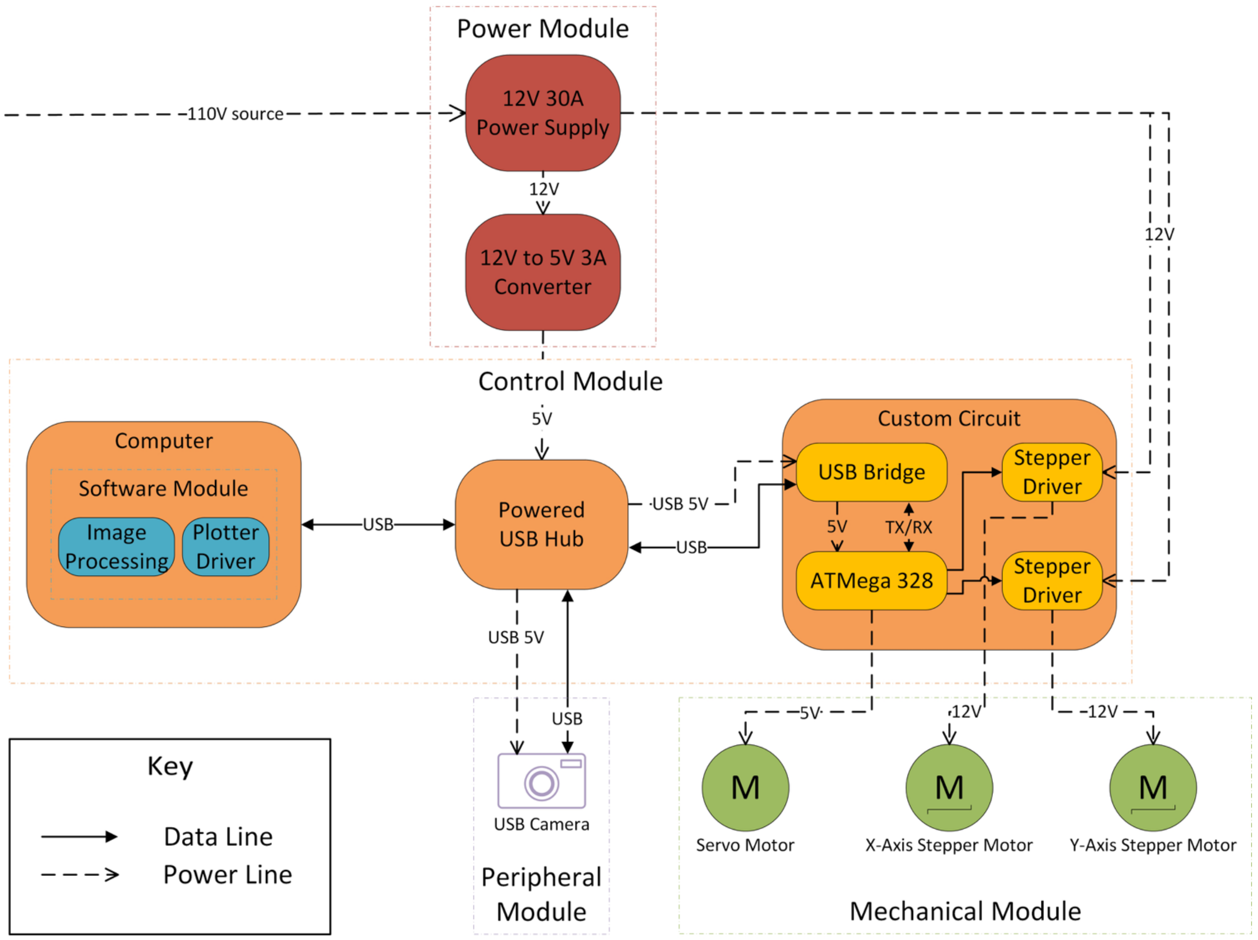


Figure 1: Block Diagram of Electronic Components

2.2 Mechanical Design

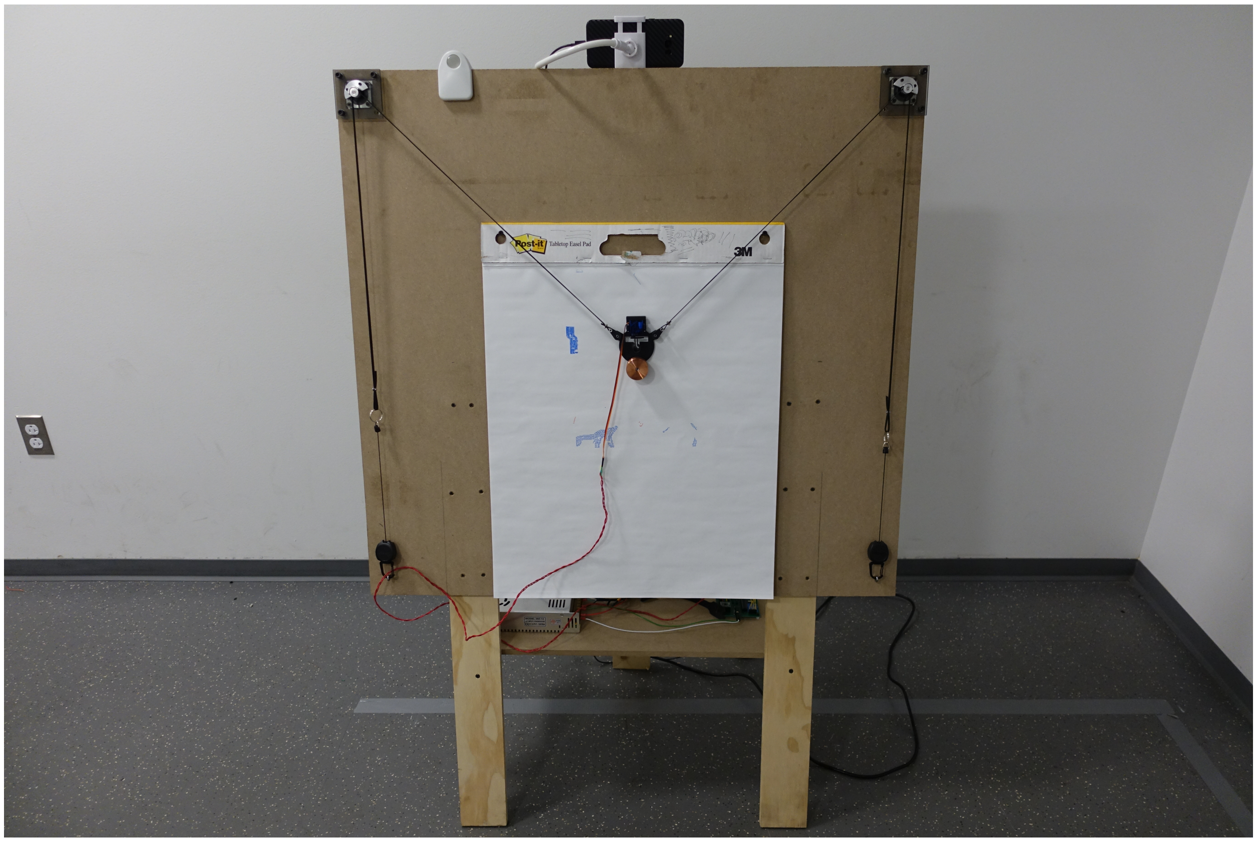


Figure 2: Complete Mechanical Design

2.2.1 Dimensions

The dimensions of the board and placement of the motors significantly impact the total drawing area and the quality of the drawn image. Too great of a change between the sent cartesian value and actual translation value implies poor resolution; a tension that is too high or too low in a given area is also problematic, as movement restriction or backlash can occur. For these reasons, calculating an optimal board size is important, as this will impact the quality of our final product and the impact on the viewer; the drawing area must also be greater than 8.5 by 11 inches, as a common printer can produce an illustration of this size.

A given cartesian point can be converted into an angle and distance and vice versa using the following relations:

$$x = cos(\alpha) * d \tag{1}$$

$$y = sin(\alpha) * d \tag{2}$$

$$d = \sqrt{x^2 + y^2} \tag{3}$$

$$\alpha = atan2(y, x) \tag{4}$$

These equations are used with the Law of Cosines to check the position of the head after a movement in the x, y coordinate system. This allows for resolution to be determined across the entire drawing surface. An unacceptable deviation is an actual movement that is greater than 1.4 times the expected movement.

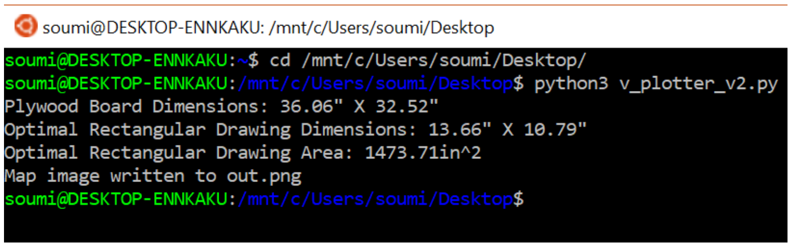
The following equations are used to calculate the tension on each belt, where t_{left} and α_{left} correspond to the tension and angle of the left belt, respectively:

$$t_{left} = \frac{\cos(\alpha_{right}) * m}{\cos(\alpha_{left}) * \sin(\alpha_{right}) + \sin(\alpha_{left}) * \cos(\alpha_{right})} \tag{5}$$

$$t_{right} = \frac{\cos(\alpha_{left}) * m}{\cos(\alpha_{left}) * \sin(\alpha_{right}) + \sin(\alpha_{left}) * \cos(\alpha_{right})} \tag{6}$$

An unacceptable belt tension is a force less than half the weight of the end effector, or greater than 1.5 times the weight of the end effector.

The source referenced for these equations contains code that performs these calculations by fixing motor positions, fixing borders, and calculating areas of low tension and accuracy [3]. We used this code and made several modifications, which allowed us to specify the size the drawing page, as well as the size of the borders. Additionally, the program now fixes the motors to the correct offsets from the corners of the board based on the NEMA17 motor dimensions. We also modified the program to calculate the largest optimal rectangular drawing area, which is where both tension and resolution are optimal [4]. Finally, we convert the values computed for the board size and optimal area to inches and output them to the console. Using this code, we were able to experiment with different sizes of boards until we found an output that suited our project's needs. The program output is shown in Figure 3b below, where the gray areas correspond to the board, white to the page, and blue to areas of low tension; the crosses indicate motor placement. The console output, Figure 3a, shows the dimensions generated by the program.


soumi@DESKTOP-ENNKAKU: /mnt/c/Users/soumi/Desktop

```
soumi@DESKTOP-ENNKAKU:~$ cd /mnt/c/Users/soumi/Desktop/
soumi@DESKTOP-ENNKAKU:/mnt/c/Users/soumi/Desktop$ python3 v_plotter_v2.py
Plywood Board Dimensions: 36.06" X 32.52"
Optimal Rectangular Drawing Dimensions: 13.66" X 10.79"
Optimal Rectangular Drawing Area: 1473.71in^2
Map image written to out.png
soumi@DESKTOP-ENNKAKU:/mnt/c/Users/soumi/Desktop$
```

(a) Program Console Output

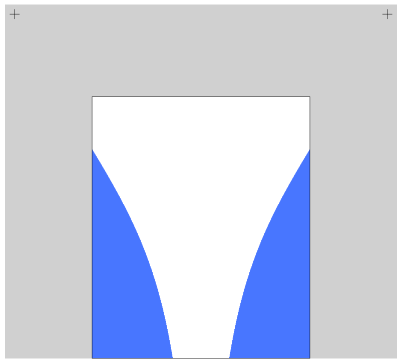

(b) Output Plot; Blue = Poor Tension

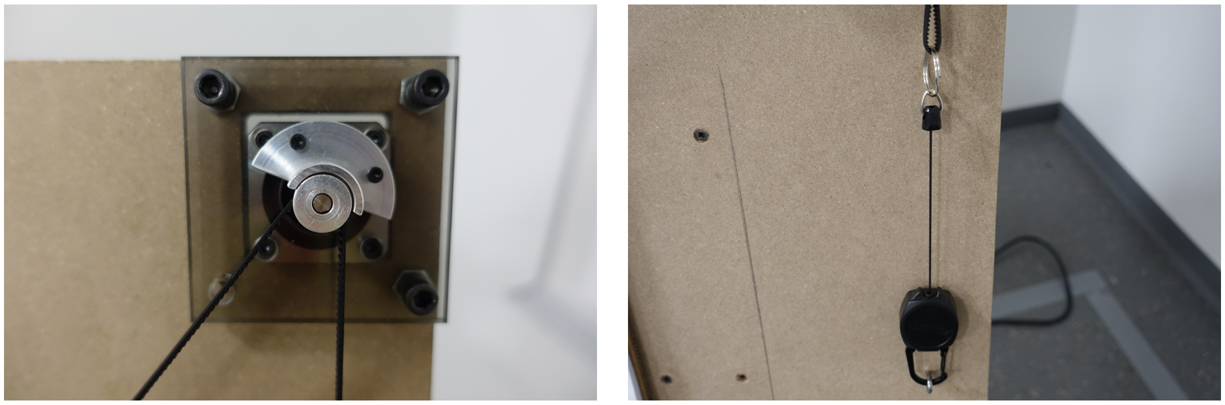
Figure 3: Complete Program Output

2.2.2 Drive System

We used GT2 belts and 16-tooth pulleys for our drive system. This allows us to precisely calculate the steps needed for a linear belt movement of one millimeter:

$$STEPS\_PER\_MM = \frac{STEPS\_REV * MICRO\_STEPS}{BELT\_PITCH * PULLEY\_TEETH} = \frac{200 * 16}{2 * 16} = 100 \tag{7}$$

Additionally, we used retracting badge holders as constant force springs to keep the belts aligned on the pulleys. Figure 4 shows the complete drive system.



(a) Pulley and Belt Configuration

(b) Belt Alignment System

Figure 4: Complete Drive System

2.2.3 End Effector

A robust end effector design is crucial for our system to work properly, as this is the piece that is driven by the motors of our project and contacts the page. Additionally, our design requires a servo to lift the pen to allow for non-drawing movements and a way to secure a pen without using screws. To begin, we looked at pre-existing end effector designs for v-plotters and used these designs to inform our own [5]. We decided to use screws and bearings to connect the belts to the end effector, as this minimizes the chances of pieces wearing out over time; our design also has mounting points for the servo and a compression lock to secure the pen. The completed end effector assembly is shown in Figure 5.

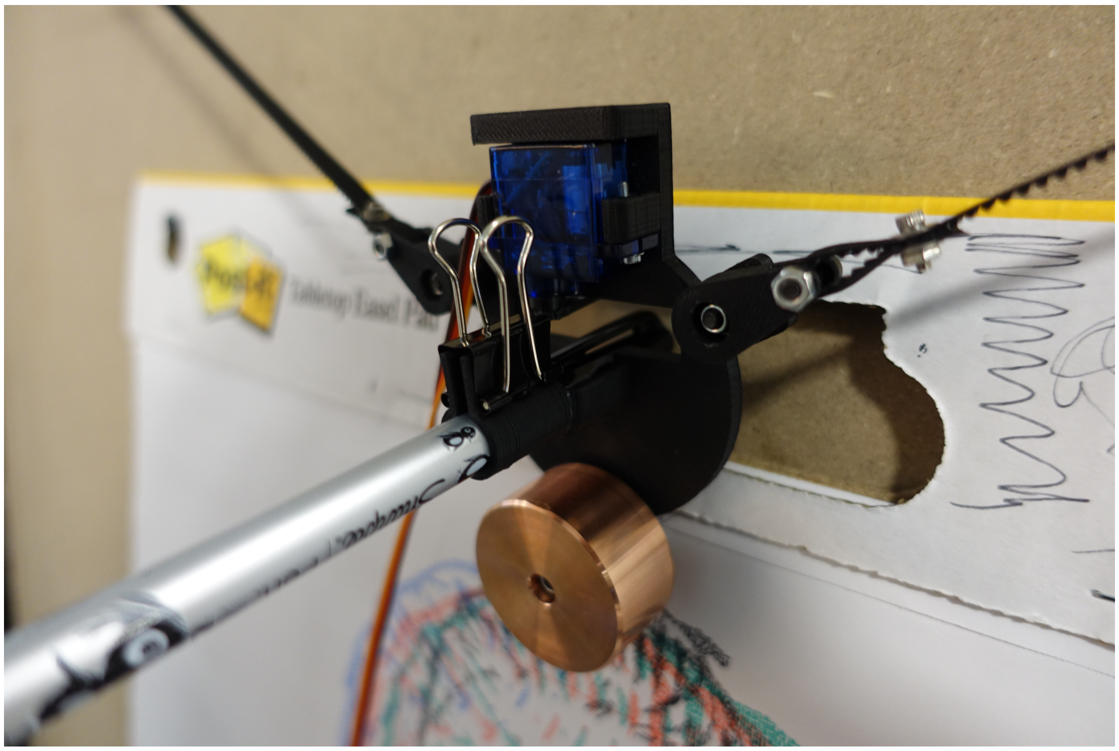


Figure 5: End Effector Assembly

2.3 Power Module

2.3.1 12V 30A Power Supply

The 12V power supply converts 120V AC mains voltage to 12V DC, which is used to power the stepper motors and the 12V to 5V converter.

2.3.2 12V to 5V Converter

The 12V to 5V converter supplies 5 volts to the Powered USB Hub.

2.4 Control Module

2.4.1 Computer

The computer is our image processing and data facilitation unit for our entire system. It allows us to receive images from the phone camera, filter the image, vectorize, and convert the output to consumable G-code for our custom circuit to read and drive the motors. This component is especially necessary to achieve the caricature effect for our drawing robot. The computer will interface, through our USB hub, with the ATmega328 and the camera. This component is necessary to generate consumable G-code and allow our robot to perform the entire process autonomously.

2.4.2 Powered USB Hub

The powered USB hub supplies 5 volts to the servo and our custom circuit; it also facilitates data transfer between the computer and the rest of the machine.

2.4.3 Circuit (PCB)

The PCB receives commands via USB, contains a microprocessor to run our firmware, and connects all components necessary to drive the mechanical system. The interface between USB and the microprocessor is handled by a Silicon Labs CP2102, an inexpensive and ubiquitous USB bridge IC [6]. We use an ATmega328 microprocessor with a 16MHz oscillator to achieve the maximum clock frequency of the device, to ensure that we accommodate our requirements for instruction throughput [7]. The in-chip serial programming (ICSP) pins are made accessible with pin headers to allow the bootloader to be burned on to the EEPROM of the ATmega328.

We use the Pololu carrier board for the Allegro A4988 stepper motor controller to drive the stepper motors [8]. The MSX pins on the IC have been wired to keep the device in 1/16 microstep mode, and the digital control inputs are wired to digital I/O pins on the ATmega.

We add one 100μF decoupling capacitor per stepper motor driver to reduce ripple from the power supply, as specified in the reference circuit in the A4988 documentation.

Appendix B shows the full schematic.

2.5 Peripheral Module

2.5.1 Camera

The camera is used to capture our subject; we used an Android phone connected via USB as our camera, with some Python code to facilitate the capture and transfer of images. This code is shown in Appendix F.

2.6 Mechanical Module

2.6.1 Stepper Motors

We are using two Kysan Nema 17 1.8°-step stepper motors in our project, with each motor corresponding to one of the plotter's axes.

2.6.2 Servo Motor

The servo motor is fitted onto the end effector; its function is to lift the end effector off the easel to allow the pen to glide to another location without dragging. The servo motor will be connected to our custom circuit.

2.7 Firmware Module

The firmware for the ATmega328 converts a sequence of G-code instructions to incremental motor movements. The firmware buffers instructions over the USB bridge, parses and executes the instructions, performs the necessary inverse kinematic calculations, executes the steps for each motor, and returns an acknowledgement signal to the computer.

2.7.1 Serial Communication

To communicate G-Code instructions, we delimited instructions with the new line character `'\n'` and created a communication control flow based on this delimiter. A single drawing is represented as a file containing delimited G-Code instructions and is sent using a Python script. The control flow for communication can be seen below:

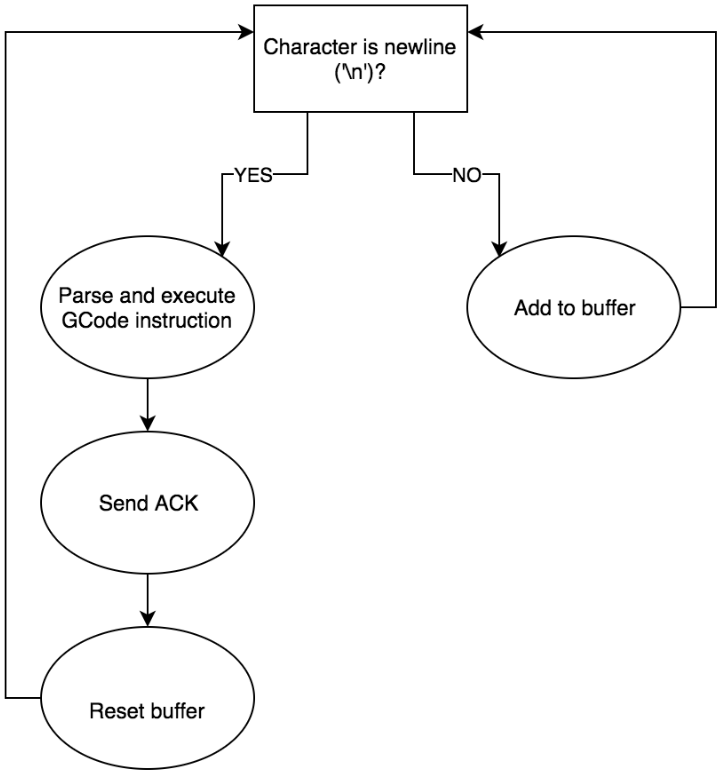

Character is newline ('\n')?
YES
NO
Parse and execute GCode instruction
Add to buffer
Send ACK
Reset buffer

Figure 6: Control Flow for Serial Communication

There is only one branch condition based on the incoming character. If the character is `'\n'`, we parse and execute the G-Code instructions. Once the instruction is successfully parsed and executed, we send an acknowledgement signal to the computer and reset the instruction buffer. If it is not `'\n'`, we simply add to our instruction buffer and wait for the next character.

2.7.2 Inverse Kinematics

The number of steps to be executed on each motor is solved using inverse kinematic equations. Given initial and final X/Y coordinates, we calculate the change in belt length, ΔR_1 and ΔR_2, which is used to calculate the number of steps for each motor, $MSteps_1$ and $MSteps_2$. An illustration of the inverse kinematics can be seen below:

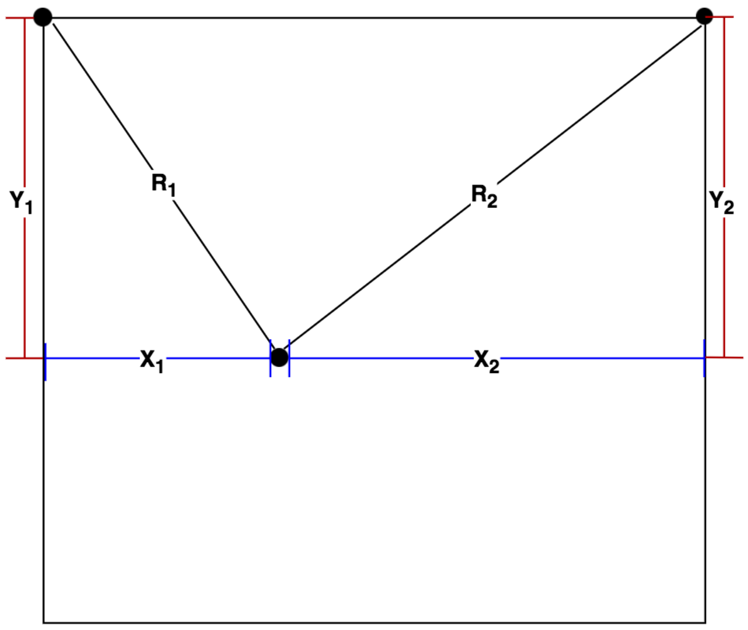


Figure 7: Diagram of inverse kinematic equations

The following equations describe the Pythagorean equations that calculate the change in belt lengths, ΔR_1 and ΔR_2. These belt lengths are then used as input to the next equations that calculate the number of steps for each motor, $MSteps_1$ and $MSteps_2$. The constant, $STEPS\_PER\_MM$, refers to the result of equation 7.

$$\Delta R_1 = \sqrt{X_{1,final}^2 + Y_{1,final}^2} - \sqrt{X_{1,initial}^2 + Y_{1,initial}^2} \tag{8}$$

$$\Delta R_2 = \sqrt{X_{2,final}^2 + Y_{2,final}^2} - \sqrt{X_{2,initial}^2 + Y_{2,initial}^2}x \tag{9}$$

$$MSteps_1 = STEPS\_PER\_MM * \Delta R_1 \tag{10}$$

$$MSteps_2 = STEPS\_PER\_MM * \Delta R_2 \tag{11}$$

2.7.3 Step Execution

After calculating the number of steps for each motor, we must properly execute these steps for each motor. This problem is posed in the context of a diagonal movement in Appendix C. The nature of our ”v-plotter” does not allow us to draw a straight line by simply extending one motor at a time. This limitation is illustrated in the following diagram:

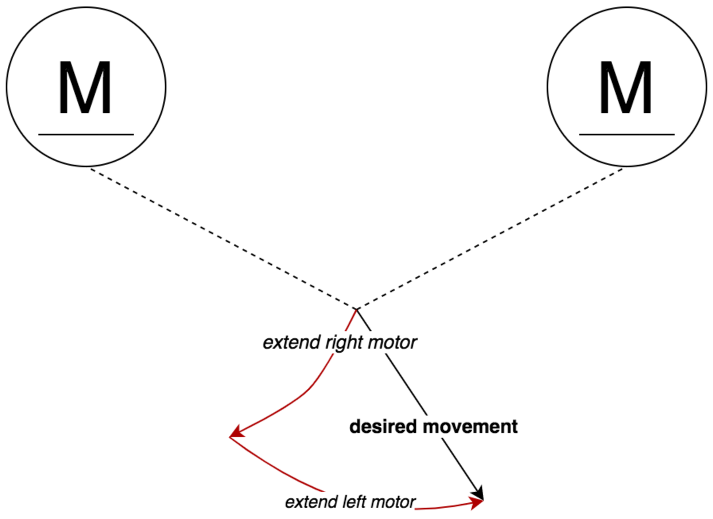
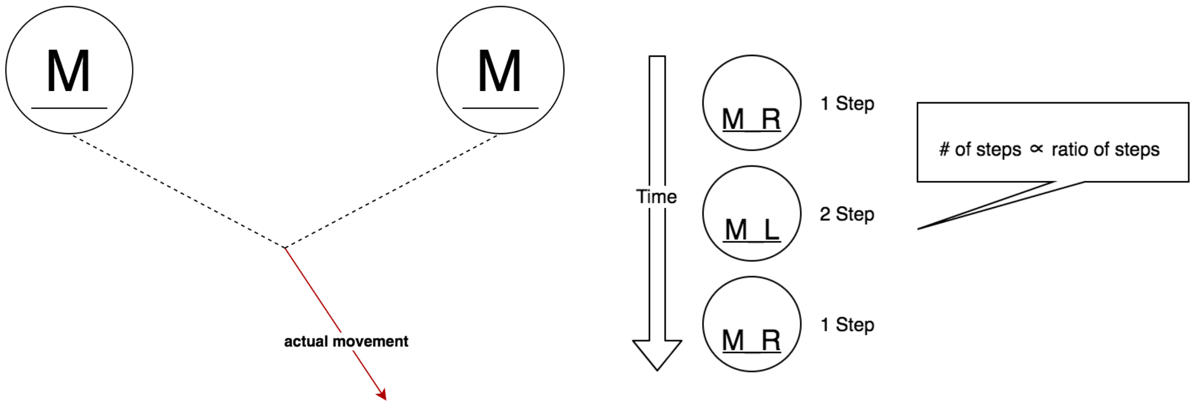


Figure 8: Limitation of completely extending the belts one at a time in a v-plotter

By extending one motor at a time, the actual movement mimics an ”L”, which incorrectly represents the desired movement. One solution is to simply drive each motor simultaneously, but since the ATmega328 is a single-core microchip, this is not possible [7]. Our initial solution interleaved the steps for each motor, where the increments are determined by the ratio of left and right motor steps. An illustration of this solution can be seen below:



(b) Interleaved Steps Over Time

(a) Correct Movement

Figure 9: Illustration of how the ”L” movement is avoided by interleaving steps

The left diagram depicts the correct movement; the right diagram shows the interleaving of motor steps over time. This solution worked, but the speed and jerkiness affected precision. These negative effects were caused by manually driving the HI and LO signals that controlled the motors. This meant that we did not account for the current acceleration of the belts and the direction in which the belt was previously traveling.

This is a problem, however, that has already been solved by an external library, so our final version uses that instead [9]. After using an external library, we found a 30% increase in speed and less jerkiness in our drawing, which noticeably increased our precision. We were also able to add a 4th parameter to our G-Code that controlled the speed of the stepper motors from 5 to 35 millimeters per second.

2.7.4 Acknowledgement Signal

Since the ATmega328's serial buffer is only 64 bytes, sending all of the G-Code instructions at once causes overflow [10]. To alleviate the small buffer size, we used the ASCII character `'*'` as an acknowledgement signal for the computer to send the next G-Code instruction. A diagram of how this system works can be seen below:

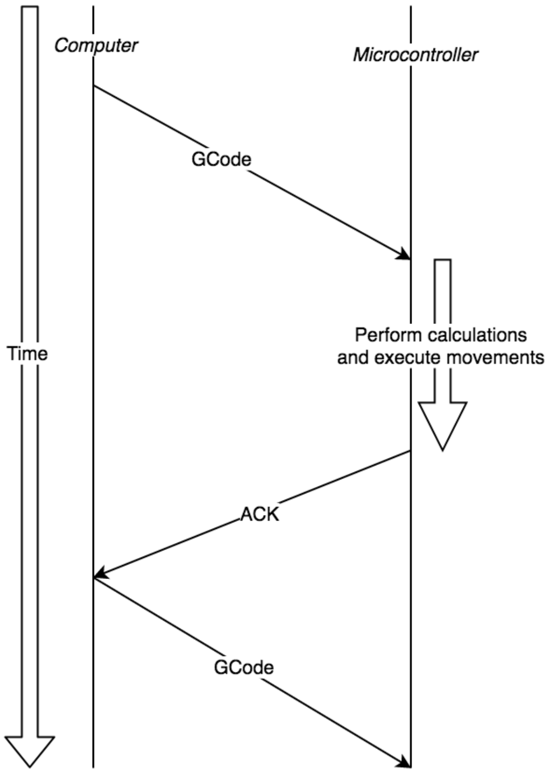


Figure 10: Diagram that illustrates the acknowledgement system

Figure 10 uses the Y-axis as time, the left side as the computer, the right side as the ATmega328, and crossing vectors as the transfer of data. The computer waits for an acknowledgement signal before sending the next instruction, effectively avoiding the truncation of G-Code instructions.

2.8 Image Processing Module

The Image Processing Module is responsible for processing the captured image and producing G-Code instructions that are sent to the machine. This software module is written in Python and uses the SciKit-Image, SciPy and OpenCV libraries. The requirements of this module are that the entire process takes less than 60 seconds on an average laptop CPU and that predicted drawing times take less than 15 minutes. A summary of the stages in this module are illustrated in Figure 11 below.

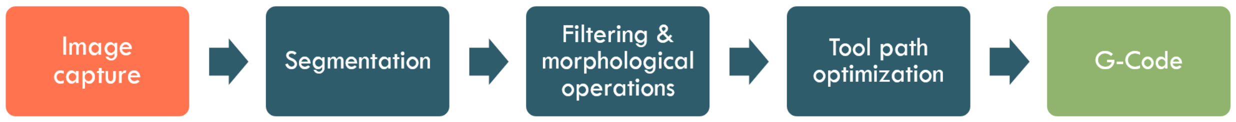


Figure 11: Image Processing Software Suite

2.8.1 Segmentation

Because our machine needs to operate under a variety of realistic settings, we cannot guarantee that the subject will be in placed in front of a uniform background. Therefore, before we can create a drawing of their face, we need to separate the face from the rest of the image. This problem has been studied before. Android phones, for example, perform face segmentation using an end-to-end convolutional neural network model. Although the results are state-of-the-art, this approach has some serious disadvantages: namely, long inference times, and reliance on a pretrained black-box model that we can not modify to our specific needs. Instead, we investigated segmentation methods based on graph cuts, which leverage image statistics and traditional graph algorithms.

For our software pipeline, we selected the GrabCut segmentation algorithm, which was first described in the 2004 paper, GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. [11]The algorithm uses Gaussian Mixture models to represent the distributions of foreground and background pixels. At each iteration, GMM parameters are updated given the current best guess for foreground and background regions. Then, an energy function that sums foreground likelihood (unary) and smoothing (pairwise) terms for every pixel is constructed, and a segmentation mask is computed with a min-cut solver. This process repeats until the segmentation converges. A reference implementation is provided available in the OpenCV library [12]. For inputs, the algorithm only requires initial guesses for foreground and background. We supply this initial guess by detecting a face in the image with Haar Cascades, which computes a bounding box around the detected face [13]. Pixels inside the bounding box are treated as probable foreground, and pixels outside the bounding box are treated as probable background. The results are visualized below.

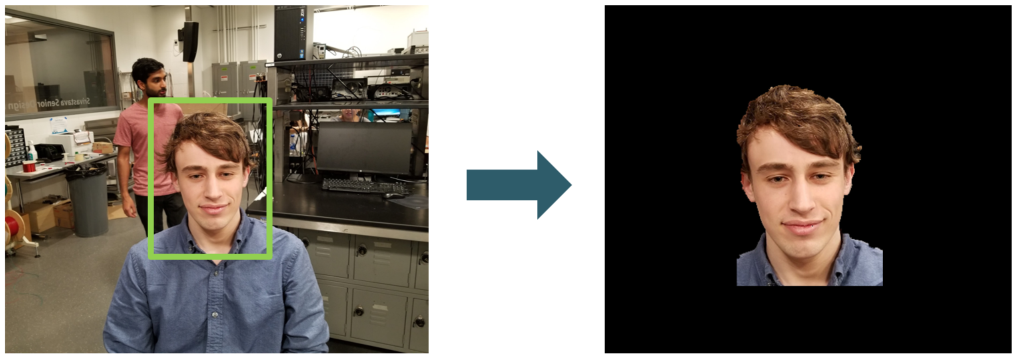


Figure 12: GrabCut segmentation algorithm results

2.8.2 Filtering and Morphological Operations

In this section we describe the filtering and morphological operators that we use to convert from a color image to a binary image that has the appearance of a hand-drawn sketch. First, we convert the RGB image to grayscale using the weighted average (luminosity) method. Then we perform edge detection. Common edge detection methods that are implemented in most image processing packages include the Canny edge detection algorithm, Sobel filtering and Laplacian of Gaussian filtering. We decided to use the Laplacian of Gaussian method because it is simple, can be performed with a single convolution operation, has a single parameter to tweak, and gives satisfactory aesthetic results. The Laplacian of Gaussian can be viewed as convolution with a blurring (Gaussian) kernel followed by convolution with a kernel that approximates the magnitude of the spatial derivative (Laplacian). Associativity of convolution allows these steps to be combined into a single operation. The full Laplacian of Gaussian filter kernel is defined by

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{12}$$

which is sampled to yield a discrete kernel [14]. Increasing sigma increases the amount of blur applied to the image before differentiation, which results in smoother images and fewer edges. We then apply a threshold to obtain binary images. For the next stage, we perform a binary erosion operation, which reduces the area of connected components in the image. Finally, we remove small components entirely from the image [15]. These operations reduce the number of filled pixels and remove small artifacts from the image, serving our objective of minimizing drawing time. The effects of these operations are visualized below.

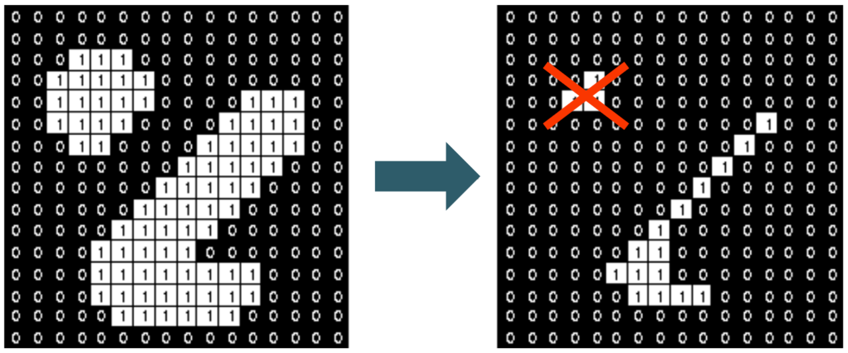


Figure 13: Morphological erosion and small object removal

2.8.3 Toolpath Optimization

There are several possible approaches to generating a toolpath for a binary image. The simplest method is to simply scan across the paper line by line from top to bottom. At a typical vertical resolution of 250 pixels, this takes considerable time (detailed analysis in Table 1). Instead, we chose to formulate the problem as an instance of the Traveling Salesman Problem, in which the end effector must touch every filled pixel (city) in the image while minimizing total tour length. Although the problem is NP-Hard, the TSP is a heavily studied problem in computer science and many excellent heuristic-based, sub-optimal algorithms exist [16].

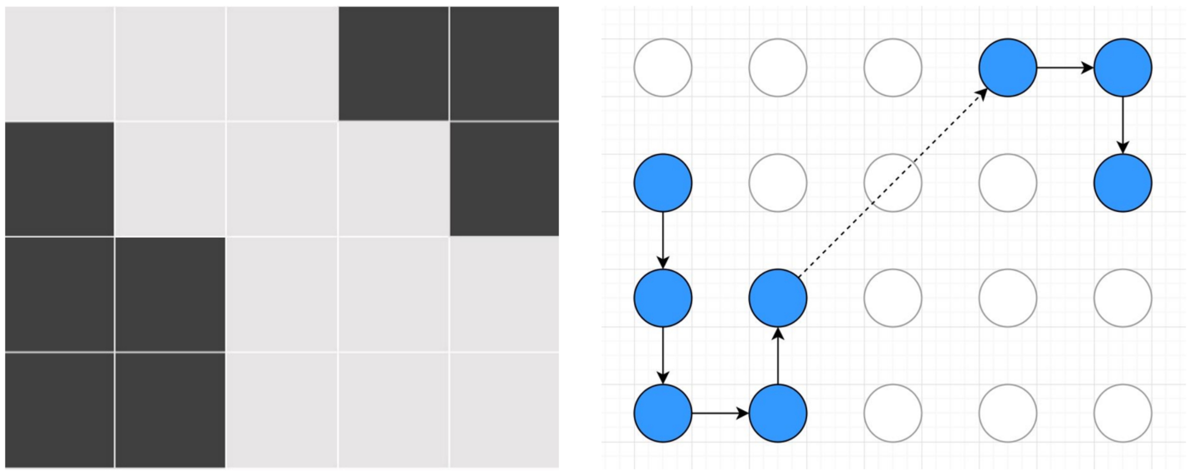


Figure 14: Binary image and its corresponding TSP instance (with solution)

The above diagram illustrates the conversion from a binary image to a toolpath using the TSP formulation. Strictly speaking, the TSP solver finds a closed path that is not oriented, but an arbitrary direction can be chosen, and the final connection omitted, as has been done in the diagram for the sake of clarity. A hypothetical solution for this toy instance is given by:

$$(2,1) \rightarrow (3,1) \rightarrow (4,1) \rightarrow (4,2) \rightarrow (3,2) \rightarrow (1,4) \rightarrow (1,5) \rightarrow (2,5)$$

The dashed line indicates that the tool retracts during the traversal of this edge. More precisely, the pen needs to retract whenever the tool moves a between two pixels that are greater than a specified Euclidean distance β apart. For $\beta = \sqrt{2}$, the pen will only draw lines between pixels that are 8-connected, and for β = 1, the pen will only draw lines between pixels that are 4-connected. Since lifting the pen adds an average of 250ms to the drawing time, we need to minimize the number of pen lifts in order to minimize drawing time; increasing β reduces the number of pen lifts by relaxing the criteria for when a lift should occur. This suggests a simple algorithm for generating a sequence of G-Code commands from a sequence of pixel coordinates, visualized in Figure 18 located in Appendix E.

As previously mentioned, the exact solution to the TSP is intractable, so we implemented a greedy algorithm using the nearest neighbor insertion heuristic. This heuristic has been shown to produce solutions that are often within 25% of the optimal solution in terms of tour length [17]. A brief summary of the algorithm is as follows: we start with a random city and then add the next closest city not already in the tour until all cities have been added. The naive implementation of this algorithm ran in $O(n^2)$ time where n is the number of cities (filled pixels), but we were able to improve this to $O(n \log(n))$ by using a KD-Tree, which supports $O(\log(n))$ nearest-neighbor queries. Since the KD-Tree implementation that we used does not support deletion, we also maintained a hash set of visited cities, and expanded the search neighborhood exponentially until a new city is found. The full pseudocode is included in Appendix D. Run times and estimated drawing times are included in the corresponding verification section.

3 Design Verification

We unit tested off-the-shelf components and verified they met our operating requirements. The verification process of these components can be found in Appendix A. The following section outlines the verification procedures for the major blocks of our design.

3.1 Mechanical Design

We verified our mechanical design during assembly. First, we wired our power supply to the mains and verified the output was within the acceptable parameters; next, we wired our circuit to the motors and verified that each axis could drive the end effector properly. We then tested 5mm lines on the X and Y axis and measured their lengths with an digital caliper. The measurements for these lines, in the good region, yielded less than 10% error shown in Appendix H. Further tests on the accuracy of our design were done through our firmware.

3.2 Firmware Module

We tested our firmware by first unit testing all four arguments to our G-Code instructions. By sending the following G-Code, we qualitatively observed the behavior of our system and confirmed that it was working correctly. The G-Code for this unit test can be found in Appendix I. After determining that our unit tests succeeded, we sent a drawing consisting of ∼6000 lines of G-Code Instructions and confirmed that it correctly drew and halted.

3.3 Control Module

We verified that our control module was operational by supplying power to the 5V and 12V inputs and attempting to move the stepper motors. We know that this circuit is working correctly when the motors resist movement, because the enable pins are high whenever the board is powered. We also unit tested pathways in our circuit by driving each motor through our firmware.

3.4 Image Processing Module

All of the major requirements of the Image Processing Module were met. The execution time of the segmentation, filtering and toolpath optimization were at most 20 seconds, 10 milliseconds and 5 seconds, respectively, for all images tested. Thus, the execution time for the entire process is well under 1 minute as required. The aesthetics results were assessed by manual inspection. Figure 15 shows some example images and the results of processing, up to and including the Filtering and Morphological Operations stage.

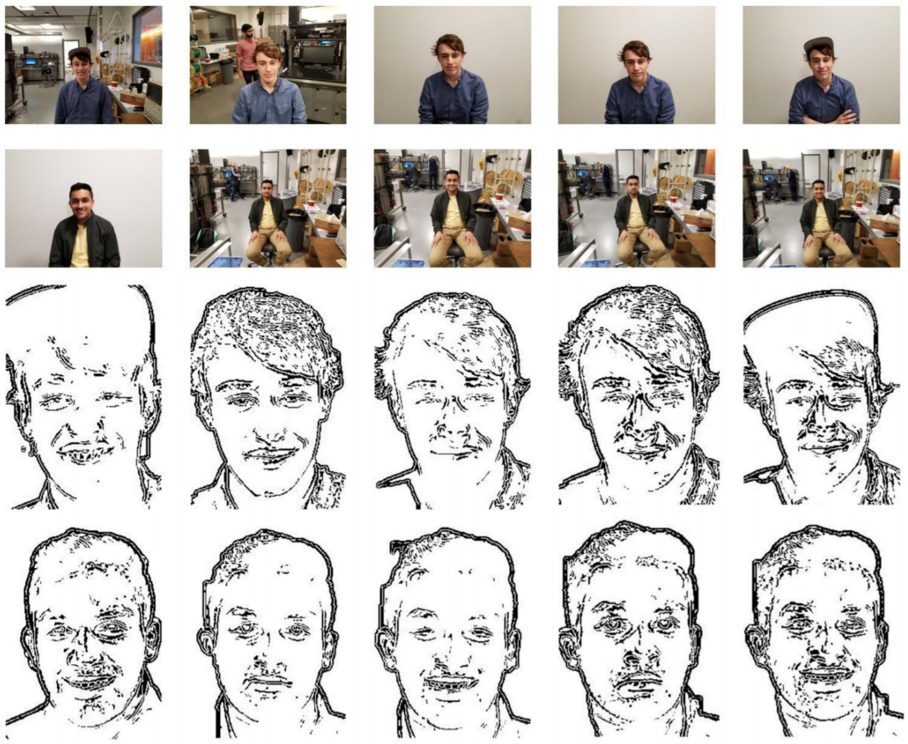


Figure 15: Example effects of filtering for 10 test images

Table 1 shows the results of toolpath optimization on a test set of $N = 10$ images. To calculate draw time, we use a measured value of 250 ms per pen lift anf the default feed rate of 25 mm per second. Greedy optimization reduces the draw time by 58% on average compared to the naive tour, bringing the total average draw time to 10.8 minutes, ±1.63. Assuming normally distributed data, the total draw time should stay within our upper limit of 15 minutes 99.5% of the time.

Table 1: Results of Toolpath Optimization

	Zig-Zag pattern	**Optimized**
Travel distance	$36,900$ ($\pm 1,780$)	$14,300$ ($\pm 2,210$)
Pen lifts	237 (± 7.10)	309 (± 42.1)
Travel time (s)	$1,480$ (± 71.3)	573 (± 88.5)
Pen lift time (s)	59.3 (± 1.77)	77.3 (± 10.5)
Total draw time (min)	25.6 (± 1.21)	10.8 (± 1.63)

4 Cost

4.1 Parts

The complete list of parts and their costs can be found in Appendix G.

4.2 Labor

We have chosen an hourly wage from a salary info sheet provided by engineering at Illinois for a Computer Engineering graduate [18].

Table 2: Labor Cost Analysis

Team Member	**Hourly Rate ($/hr)**	**Hours (hrs)**	**Cost x 2.5 ($)**
Dylan Huang	39.30	180	17685
Peter Kuimelis	39.30	180	17,685
Soumithri Bala	39.30	180	17,685
		Total	53,055

5 Conclusion

5.1 Accomplishments

We successfully assembled the robot before the deadline and were able to demo the entire pipeline from image capture to finished drawing. It takes one single command to execute the entire process and $\sim$15 minutes to complete. A finished product can be seen in Appendix J.

5.2 Ethical considerations

We considered two ethical issues while working on our project. Since the three members of this group were the primary testing subjects during development, there is a high potential for our image processing software to be over-fitted to our features. According to section 7.8.8 in the IEEE Code Of Ethics, we must treat all persons fairly and not engage in acts of discrimination [19]. To achieve this goal, we would need to do extensive testing on a diverse set of subjects to make this product enjoyable for everyone.

Finally, we wish to be realistic in our claims of what this project can do; we do not claim that our project is true art, as artists spend many hours perfecting their craft. Instead, we view our project as a fun way to introduce people to computing and robotics.

5.3 Future work

5.3.1 Mechanical Design

Several improvements could be made to the mechanical design. The assembly could be made significantly lighter and more portable; using a thinner frame, for example, may allow for the device to be collapsed and transported more easily. The end effector could also be redesigned to support multiple pen holders, which would allow for a greater range of drawing styles.

5.3.2 Firmware Module

Our communication system has no reliability protocol that ensures robust data transfer. A future iteration of the acknowledgement system could implement a TCP-like sequence numbering reliability protocol to ensure that the correct instruction is being sent [20].

References

[1] "How to Become a Caricaturist," Web page, The Art Career Project. [Online]. Available: https://www.theartcareerproject.com/become/caricaturist/

[2] S. Brennan, "The Caricature Generator," PDF, MIT. [Online]. Available: https://www.jstor.org/stable/pdf/1578048.pdf?refreqid=excelsior%3A560aca3eb8cd44c183832869c631a2fe

[3] B. Rasmussen, "V Plotter Design," Web page. [Online]. Available: http://2e5.com/plotter/V/design/

[4] "Maximal Rectangle," Web page. [Online]. Available: https://github.com/boyw165/algorithm-challenge-python/tree/master/maximal-rectangle

[5] "Vertical Plotter Gondola v3," Web page. [Online]. Available: https://www.thingiverse.com/thing:472573

[6] "CP-2012 datasheet," PDF, Silicon Labs. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[7] "ATmega328P," Web page, Microchip. [Online]. Available: http://www.microchip.com/wwwproducts/en/ATmega328p

[8] "Pololu A4988 datasheet," PDF, Pololu Labs. [Online]. Available: http://ww1.microchip.cm/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[9] "AccelStepper library for Arduino," Web page, AirSpayce. [Online]. Available: http://www.airspayce.com/mikem/arduino/AccelStepper

[10] "Expanding Arduino Serial Port Buffer Size," Web page, Internet of Home Things. [Online]. Available: https://internetofhomethings.com/homethings/?p=927

[11] "Grabcut: Interactive foreground extraction using iterated graph cuts," Article, Microsoft Research UK, 2004. [Online]. Available: https://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf

[12] "Grabcut segmentation," Web page, OpenCV. [Online]. Available: https://docs.opencv.org/trunk/d8/d83/tutorial_py_grabcut.html

[13] "Haar cascades face detection," Web page, OpenCV. [Online]. Available: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

[14] "Laplacian of gaussian operator," Web page, University of Edinburgh. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[15] "Morphological image processing," Web page, University of Auckland. [Online]. Available: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

[16] S. Skiena, *The Algorithm Design Manual*, New York, NY: Springer, 1997.

[17] "Heuristics for the traveling salesman problem," Web page, Linkoping University. [Online]. Available: https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf

[18] "Salary Info Sheet," PDF, Engineering at Illinois. [Online]. Available: https://engineering.illinois.edu/documents/Salary.Info.Sheet.pdf

[19] "IEEE Code of Ethics," Web page. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[20] S. K. Davu, “A Survey on Next Generation Tcp Derivatives for Mobile Environment,” Web page, Kent State University. [Online]. Available: http://www.medianet.kent.edu/surveys/DR03S-mobiletransport/index.html

Appendix A Requirement and Verification Table

Table 3: System Requirements and Verification

Component	Requirement	Verification	Verification status (Y or N)
12V 30A Power Supply	1. Converts wall voltage to (110V) to 12V 30A DC ± 5%	1. Use a multimeter and measure 12V/30A out of the power supply	Y
12V to 5V Converter	1. Supply a steady 5V at 3A ± 5% to the Powered USB hub	1. Measure the open-circuit voltage of the input and output of the 12V to 5V 3A converter using a multimeter 2. Measure the 3A output of the 12V to 5V 3A converter using an ammeter in series	Y
Computer	1. Receive an image from the camera 2. Apply filters to the image, save as a binary raster image 3. Convert binary raster image to G-Code	1. Ensure images taken on camera appear in specified location in the Raspberry Pi's file system 2. Open the image and verify that the filters have produced the desired result 3. Run the G-code through a toolpath simulator that checks for syntax errors and verify that the output of the G-code simulation matches the binary image file 4. Send G-code instructions to ATmega328 circuit and check for acknowledgement signal	Y
Powered USB Hub	1. 5V to 3A ± 5%	1. Measure the open circuit voltage across the VCC and GND pins of the USB female connectors and ensure 5V 3A ± 5%	Y
ATmega PCB	1. V5 (name of power input line) remains at 5V ± %10 throughout operation of machine	1. Use multimeter to measure V5 node and measure the maximum deviation	Y

Continued on next page

Table 3 – continued from previous page

Component	Requirement	Verification	Verification status (Y or N)
Camera	1. Minimum of 8-megapixel still photos 2. Automatic focusing in range of lighting conditions	1. Check dimensions of captured image 2. Empirically verify the photos appear focused and properly lit	Y
Stepper Motors	1. Drawing precision for 5 units of distance at less than 1 of angle error 2. Temperature cannot reach over 40C when operating	1. Direct the system to draw a straight line of 5 units of distance and measure with dial calibers the actual Y and X coordinates. Compare these measurements with the expected X and Y coordinates and calculate error . Ensure that error is less than 1 2. Execute the entire pipeline for a filtered image and measure the temperature during operation with an IR thermometer. Ensure that the temperature does not exceed 40C	Y
Servo Motor	1. Have enough torque to lift the end effector off the easel such that the pen is not touching the paper	1. Place the servo motor under the end effector unit. Use a function generator to supply a PWM wave with a frequency of 50Hz and a duty cycle of 7.5% for position 0. Then change the duty cycle to 10% to move the to the 90 position. Ensure that the servo motor can lift the end effector off the ground.	Y

Continued on next page

Table 3 – continued from previous page

Component	Requirement	Verification	Verification status (Y or N)
Image Processing	1. Face detection accuracy 90% or better under favorable lighting conditions 2. Background of image is masked/removed 3. All processing takes less than 1 minute.	1. Place subject 5 feet in front of camera in a uniformly lit indoor space. Capture image; repeat N=100 times. Software must correctly identify and crop face at least 90/100 times. 2. View preview image for a large (N=100) batch of test images. 3. Use a timer library to measure the wall-clock time from beginning to end process (raw image to G-code).	Y
Firmware	1. Receive G-code commands from computer 2. Convert G-code into multiple delta movements 3. Send commands to stepper drivers	1. Add log statements to print received commands; verify received command is the same as sent command 2. Execute test instruction, measure rotation of stepper motors and verify with inverse kinematic equation.	Y

Appendix B Schematic

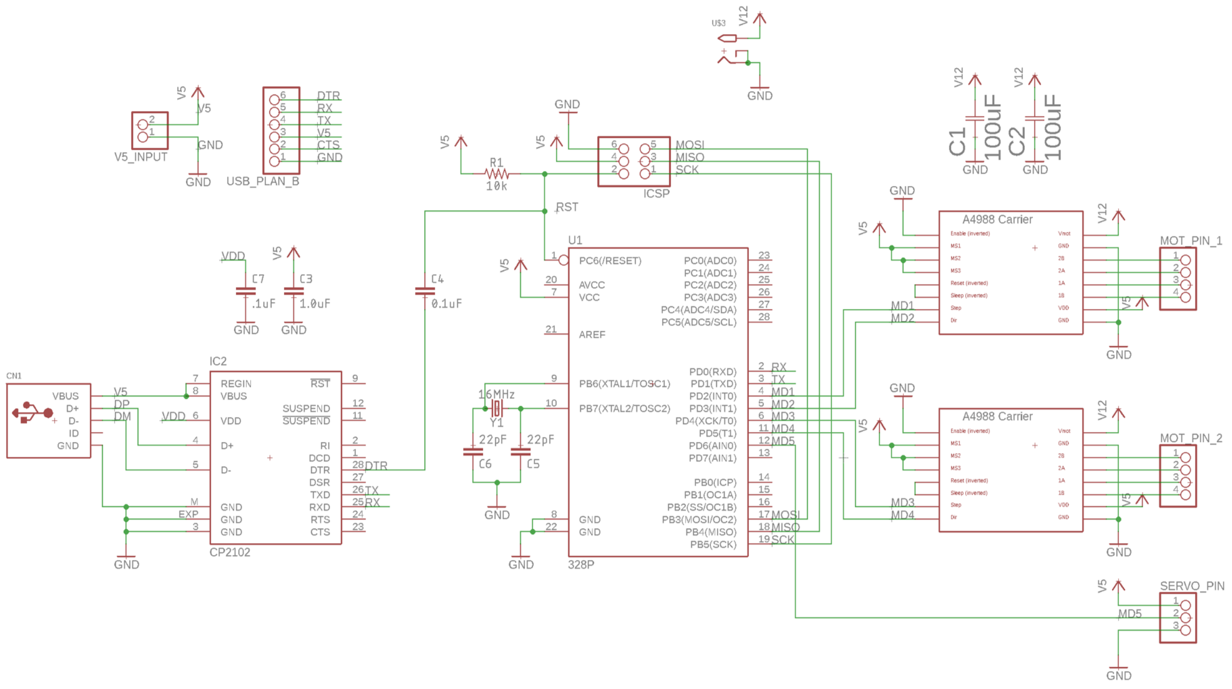


Figure 16: Circuit Schematic

Appendix C Step Execution

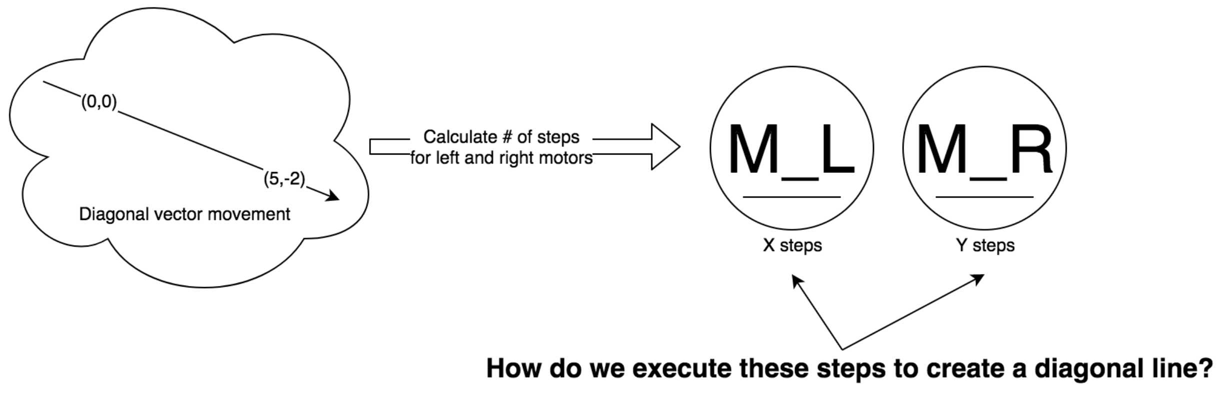

(0,0)
(5,-2)
Diagonal vector movement
Calculate # of steps for left and right motors
M_L
M_R
X steps
Y steps
How do we execute these steps to create a diagonal line?

Figure 17: Step execution problem posed in the context of a diagonal vector movement

Appendix D Pseudocode

Algorithm 1 Greedy TSP solver

```
procedure GreedyTSP($cities, N$)
    build KD tree
    $tour \leftarrow$ empty list
    $visited \leftarrow \emptyset$
    $i \leftarrow 0$
    $tour[i] \leftarrow$ random city in $cities$
    while $i < N$ do
        $k \leftarrow 2$
        $searching \leftarrow$ True
        while $searching$ is True do
            $k\_nearest \leftarrow tree.query(tour[i], k)$
            for $city \in k\_nearest$ do
                if $city \notin visited$ then
                    $tour[i] \leftarrow city$
                    $visited \leftarrow visited \cup \{city\}$
                    $searching \leftarrow$ False
                    break
            $k \leftarrow k * 2$
    return $tour$
```

Appendix E G-Code Generation Flow Diagram

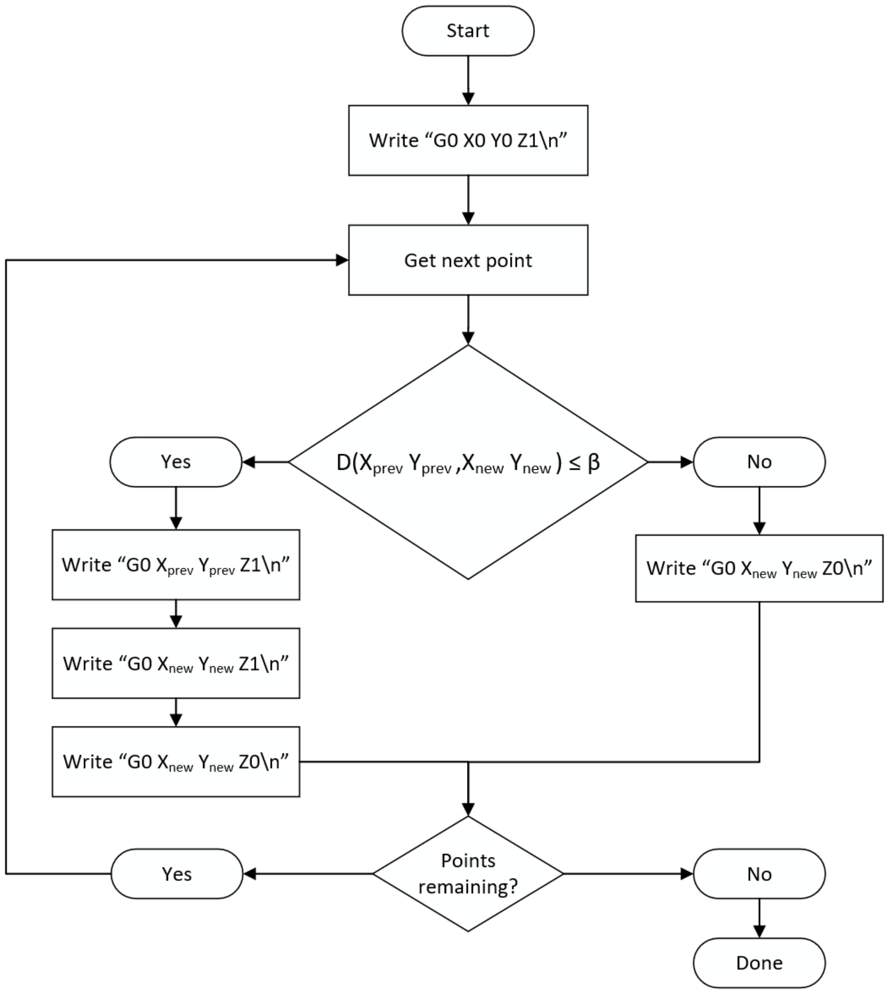
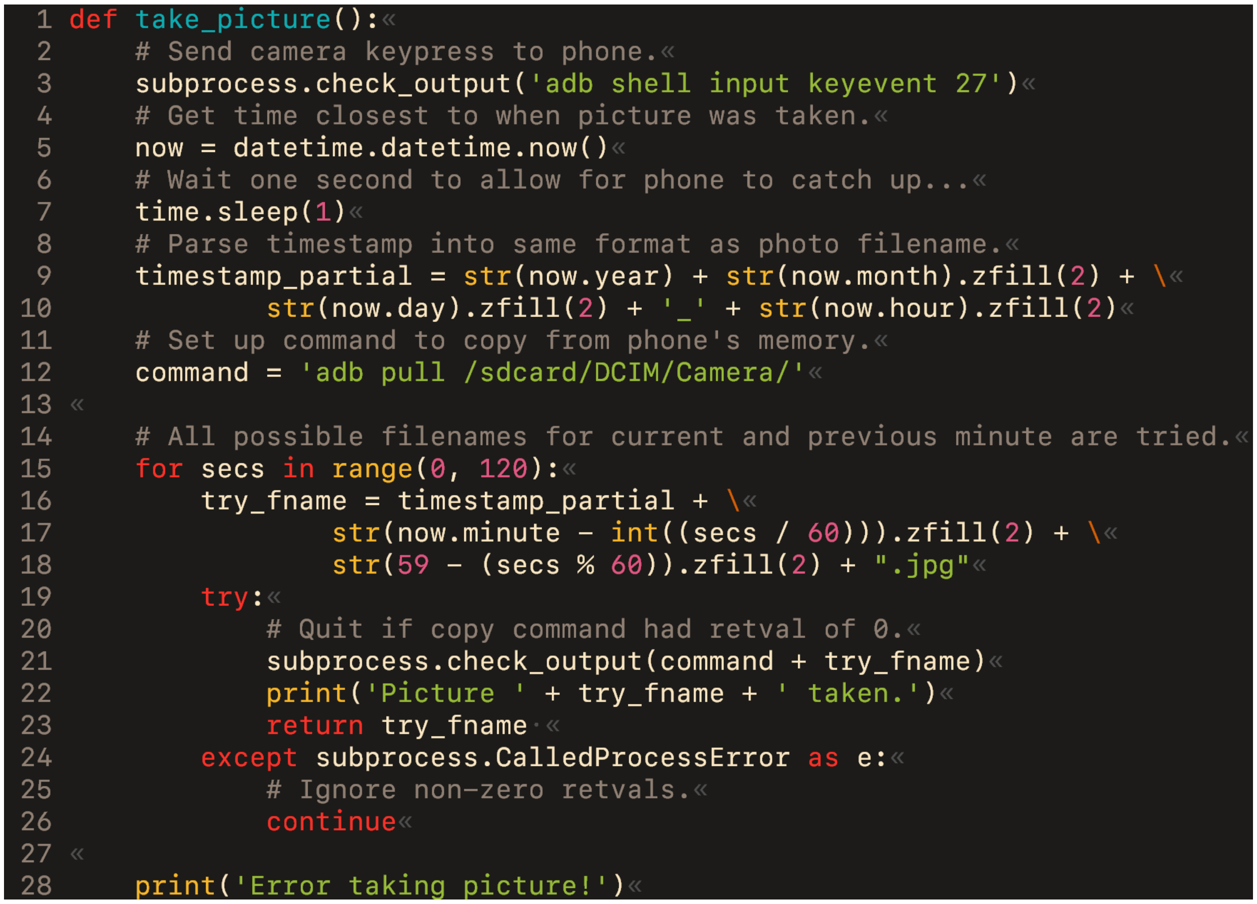

Start
Write "G0 X0 Y0 Z1\n"
Get next point
D(Xprev Yprev ,Xnew Ynew) ≤ β
Yes
No
Write "G0 Xprev Yprev Z1\n"
Write "G0 Xnew Ynew Z1\n"
Write "G0 Xnew Ynew Z0\n"
Write "G0 Xnew Ynew Z0\n"
Points remaining?
Yes
No
Done

Figure 18: Generating G-Code from a sequence of coordinates

Appendix F Camera Capture Code



```python
def take_picture():
    # Send camera keypress to phone.
    subprocess.check_output('adb shell input keyevent 27')
    # Get time closest to when picture was taken.
    now = datetime.datetime.now()
    # Wait one second to allow for phone to catch up...
    time.sleep(1)
    # Parse timestamp into same format as photo filename.
    timestamp_partial = str(now.year) + str(now.month).zfill(2) + \
            str(now.day).zfill(2) + '_' + str(now.hour).zfill(2)
    # Set up command to copy from phone's memory.
    command = 'adb pull /sdcard/DCIM/Camera/'

    # All possible filenames for current and previous minute are tried.
    for secs in range(0, 120):
        try_fname = timestamp_partial + \
                str(now.minute - int((secs / 60))).zfill(2) + \
                str(59 - (secs % 60)).zfill(2) + ".jpg"
        try:
            # Quit if copy command had retval of 0.
            subprocess.check_output(command + try_fname)
            print('Picture ' + try_fname + ' taken.')
            return try_fname
        except subprocess.CalledProcessError as e:
            # Ignore non-zero retvals.
            continue

    print('Error taking picture!')
```

Figure 19: Python code for capturing image on Android phone

Appendix G Parts Cost

Table 4: Parts Costs

Part	Part Number	Manufacturer	Unit Cost ($)	Qty	Total Cost ($)
Drawing Pad		Post-it	20.33	1	20.33
Stepper Motor		OMC Corporation Limited	13.99	2	27.98
Stepper Driver (5PK)		BIQU	8.99	1	8.99
12V-5V Converter		Kimdrox	7.99	1	7.99
Arduino Knockoff		Elegoo	10.90	1	10.90
12V PSU		BMOUO	18.56	1	18.56
USB Serial		WINGONEER	8.99	1	8.99

Continued on next page

Table 4 – continued from previous page

Part	**Part Number**	**Manufacturer**	**Unit Cost ($)**	**Qty**	**Total Cost ($)**
USB Extension		Amazon	5.34	1	5.34
Timing Belt		Mercurry	8.88	1	8.88
Pulleys		DROK	11.89	1	11.89
Extension Cord		Micro Connectors Inc.	2.73	1	2.73
Belt Holders		Key-Bak	12.45	1	12.45
Servo		ElectroBot	7.99	1	7.99
Bearings		Amico	4.21	1	4.21
Phone Mount		Aduro	10.99	1	10.99
Powered USB Hub		Plugable	16.95	1	16.95
USB to UART	336-3694-ND	Silicon Labs	1.40	3	4.20
22pF Capacitor	445-11133-1-ND	TDK Corporation	0.07	10	0.71
100F Capacitor	399-6726-1-ND	KEMET	0.43	4	1.72
USB mini B	609-4700-1-ND	Amphenol FCI	0.71	2	1.42
16MHz Crystal	X176-ND	ECS Inc.	0.69	2	1.38
Power Barrel Jack	CP-002AHPJCT-ND	CUI Inc.	1.44	1	1.44
Power Barrel Connector	CP3-1000-ND	CUI Inc.	1.04	1	1.04
Header Female	S7002-ND	Sullins Connector Solutions	0.45	4	1.80
Header Female	S7001-ND	Sullins Connector Solutions	0.42	3	1.26
Header Female	S7006-ND	Sullins Connector Solutions	0.65	6	3.90
ATmega328 Socket	ED3050-5-ND	On Shore Technology Inc.	0.33	2	0.66
Header Female	S7004-ND	Sullins Connector Solutions	0.52	2	1.04
Total					205.74

Appendix H Design Verification

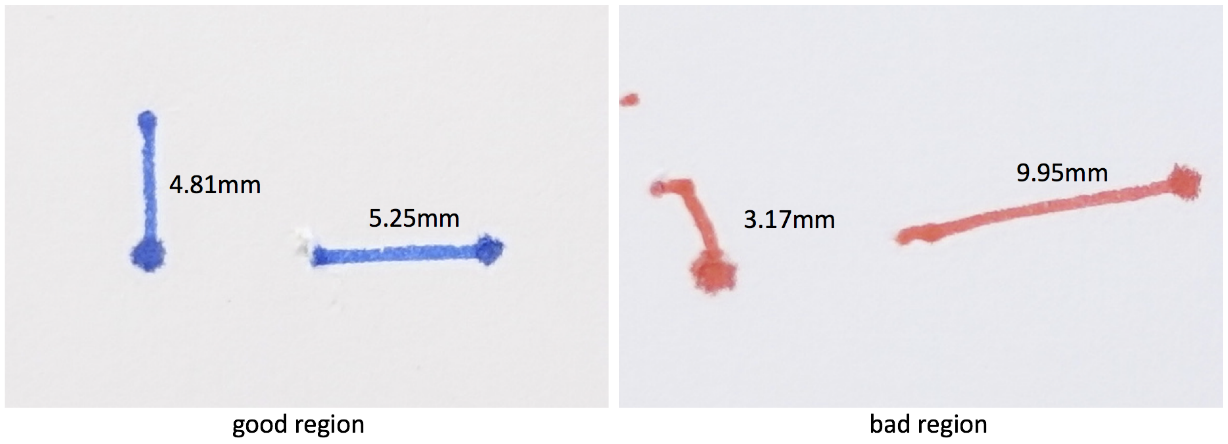


Figure 20: Testing 5mm lines on the Y and X Axis

Appendix I G-Code Unit Test

```
G0 X0 Y0 Z1 # Testing Z-Axis (Servo)
G0 X0 Y0 Z0
G0 X0 Y-50 Z0 # Testing Y-Axis
G0 X0 Y-25 Z0
G0 X25 Y-25 Z0 # Testing X-Axis
G0 X-25 Y-25 Z0
G0 X25 Y-25 Z0 F35 # Testing Stepper Motor adjustable speed
G0 X-25 Y-25 Z0 F5
G0 X0 Y0 Z0 # Reset
```

Figure 21: Firmware Verification Code

Appendix J Completed Output

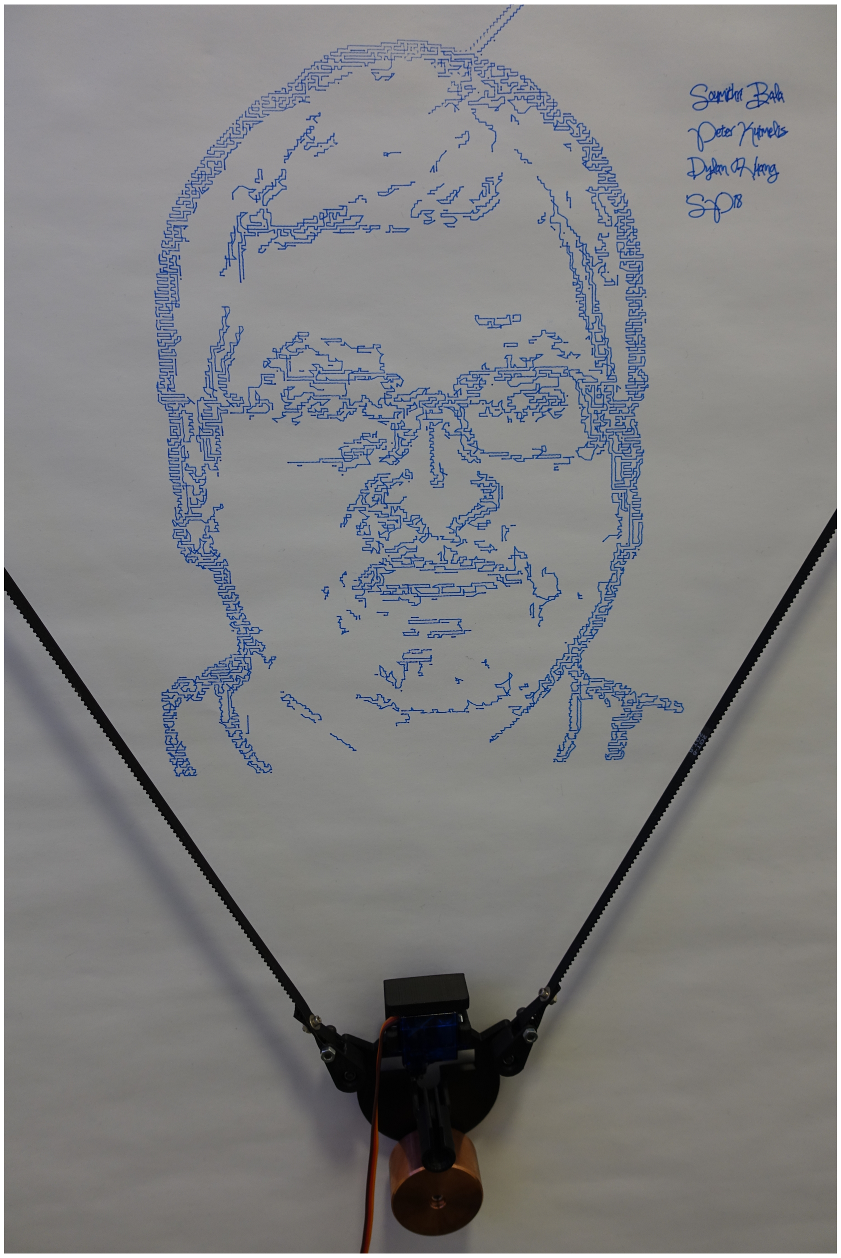


Figure 22: Completed Drawing