Developer Guide

The spydercam is designed to move a tool, such as a writing utensil, to specific points in three dimensional space. This is accomplished by feeding G-code instructions to an arduino, which processes the command and enables motors to change the amount of string that is suspending the tool from three pylons. Additional systems were also implemented that use computer vision to translate an image into gcode to have the spydercam draw it, and a system to take a photo of a line and then measure its length.

Specification	**Range of Accepted Values**
Motor Driver Power Supply	Max Supply: 36V System designed/optimised for: 12V
Arduino Nano Every USB Connection	Power Supplied: 5V USB Power Serial Connection: 9600 Baud rate
Operating Temperature	-40°C to 125°C

Figure 1: Electrical Specifications Table

User guide:

Set up steps

1. Plug arduino to computer using usb cable
2. Upload arduino code to the arduino using arduino IDE

Useage

1. Manual entry
    a. Open arduino IDE and establish serial communication with arduino
    b. Use serial monitor tool to send commands one at a time
2. Entry from gcode file
    a. Run command python script, and include the name of the file that contains the Gcode as a command line argument.

Using image processing

1. Run the image processing software.
2. When prompted type in the file name or file path of the image to be processed.

Using camera measurement

1. Attach camera mount (M6 Command)
2. Move camera to proper positioning
3. Flash camera code
4. Take photo
5. Flash motor code
6. Unmount camera (M6 command)
7. Remove SD card and insert into computer
8. Process photo using computer vision code to get line length

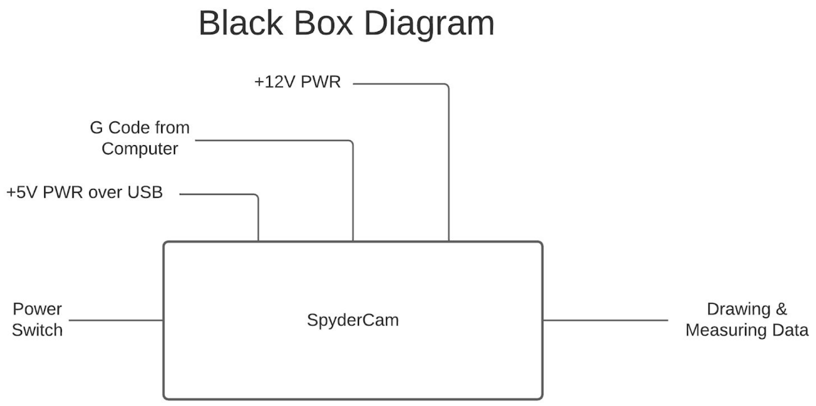

Figure 2: Block Diagram

This black box diagram shows the external inputs and outputs of the system as a whole. The inputs include 5V power and data over USB, and 12V power, and user input when using the software. The system output is movement and drawing.

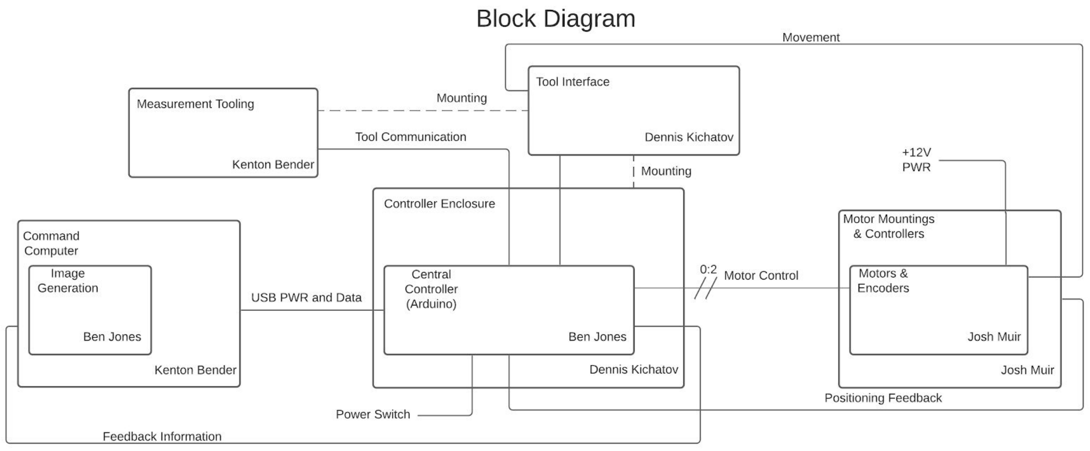

Figure 3: Block Diagram

(we should improve these)

This block diagram shows the internal inputs and outputs for the system, and how some of the blocks interact with each other. The main block is the arduino, as it acts as a central point for the other blocks, taking information from the computer and then translating it to the motors.

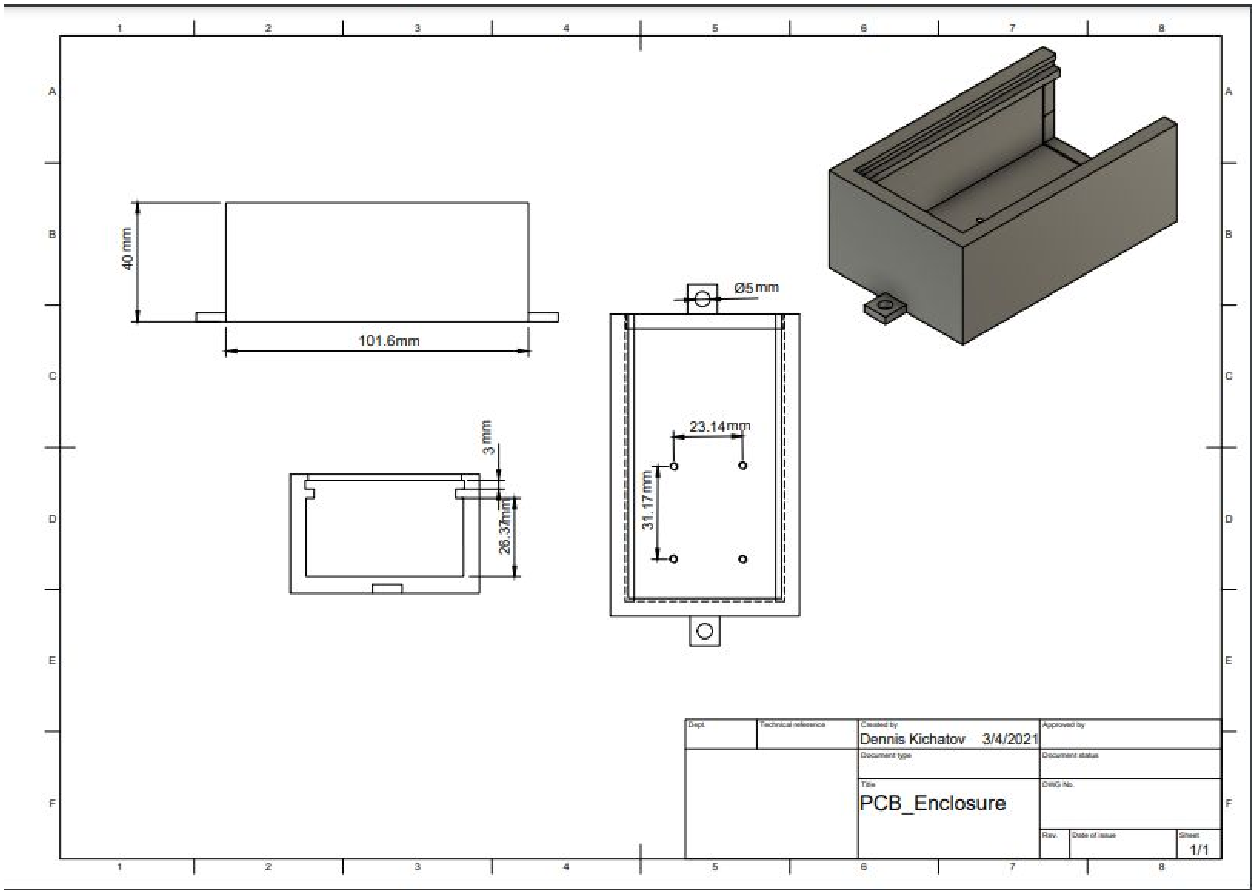

Figure 4: Enclosure Model

This is the 3D model of the team's enclosure. The Enclosure was designed to hold a PCB, which will hold our Arduino Every. The holds on the bottom of the enclosure are 2 millimeters in diameter and are designed for M2 screws. This way the PCB and Arduino will be held in place. The lid is 3 millimeters thick tile that will slide into the top half of the enclosure. The lid also has a 6mm guard on it that will close off a part of the entrance to keep the wires in place. The Enclosure is also designed to screw into the main board to ensure that it won't fall off the edge. However in case it does, the enclosure is strong enough to handle the impact.

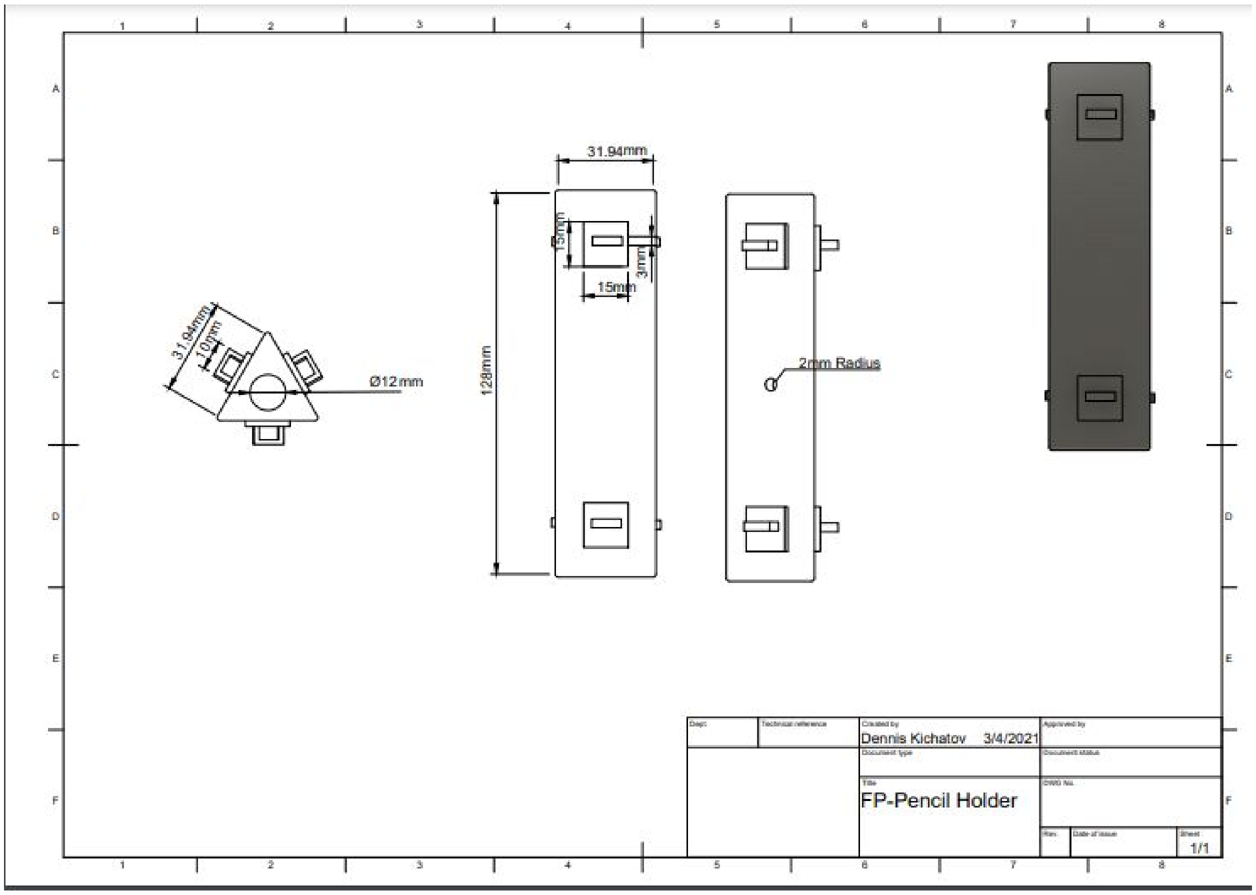

Figure 5: Mount Model

This is the 3D model of the writing utensil mount. The mount was designed to be able to hold any pencil and most types of pens. The hollow tube is in the shape of a triangle to make the tube faces face the pylons. The triangle shape also helps with the sturdiness of the mount. The hole going through the tube is twelve millimeters in diameter and is big enough to fit pencils and pens, without their caps. The hole on the side is 4 millimeters in diameter and is designed to fit a M5 screw. This screw holds the writing utensil in place. The handle bards on the side of the mount are for the string. The two handles will each have their own string that will connect back to the main string on the pulley to form a triangle. This way, the mount will be the most stable while moving.

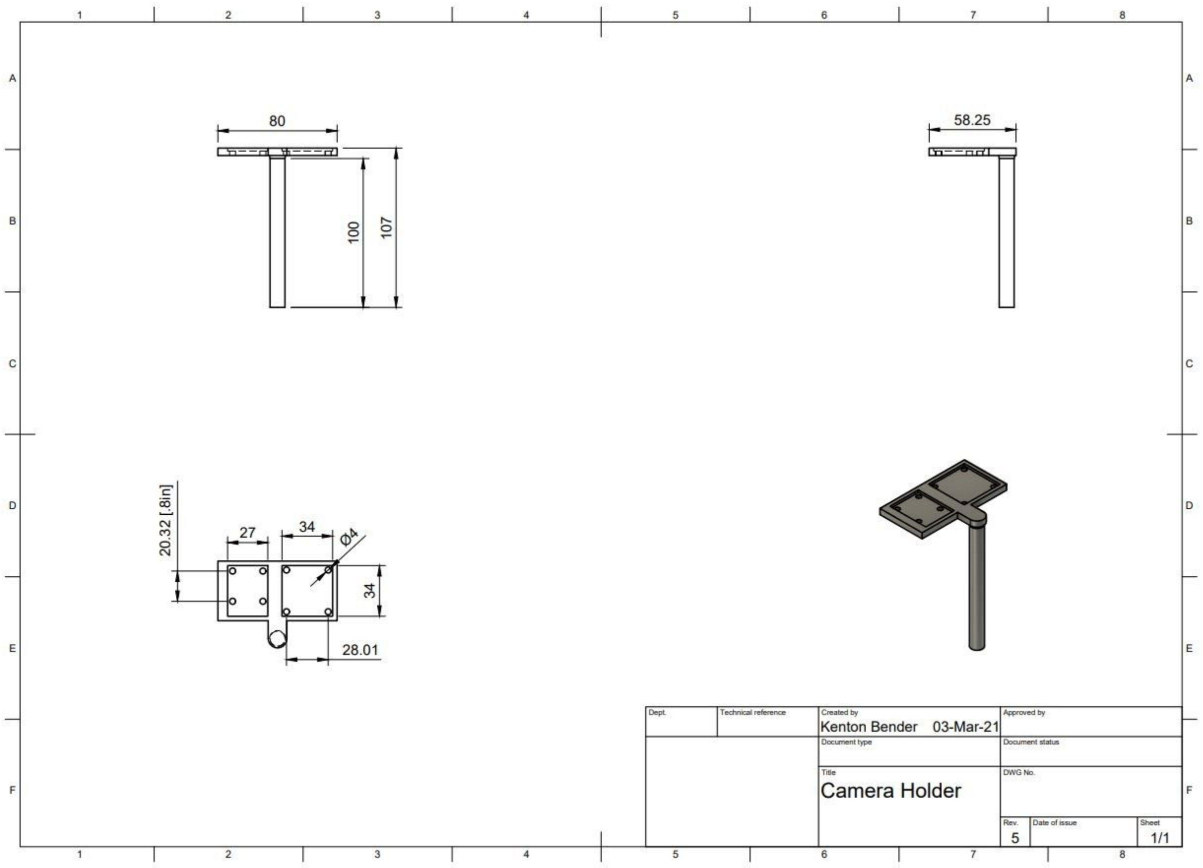

Figure 6: Camera Holder Model

This is a model of the camera holder. It is designed to mount inside of the tool mount using the long spindle coming out of it, and hold the camera facing down towards the page. The camera then takes a picture of the page and saves it to an SD card, which is removed to process the image and measure lines. There are two recessed areas on the mounting surface in which the two PCBs that hold the camera and the SD card reader breakout board can be attached. Screw holes are also aligned with holes in the PCBs to pass screws through.

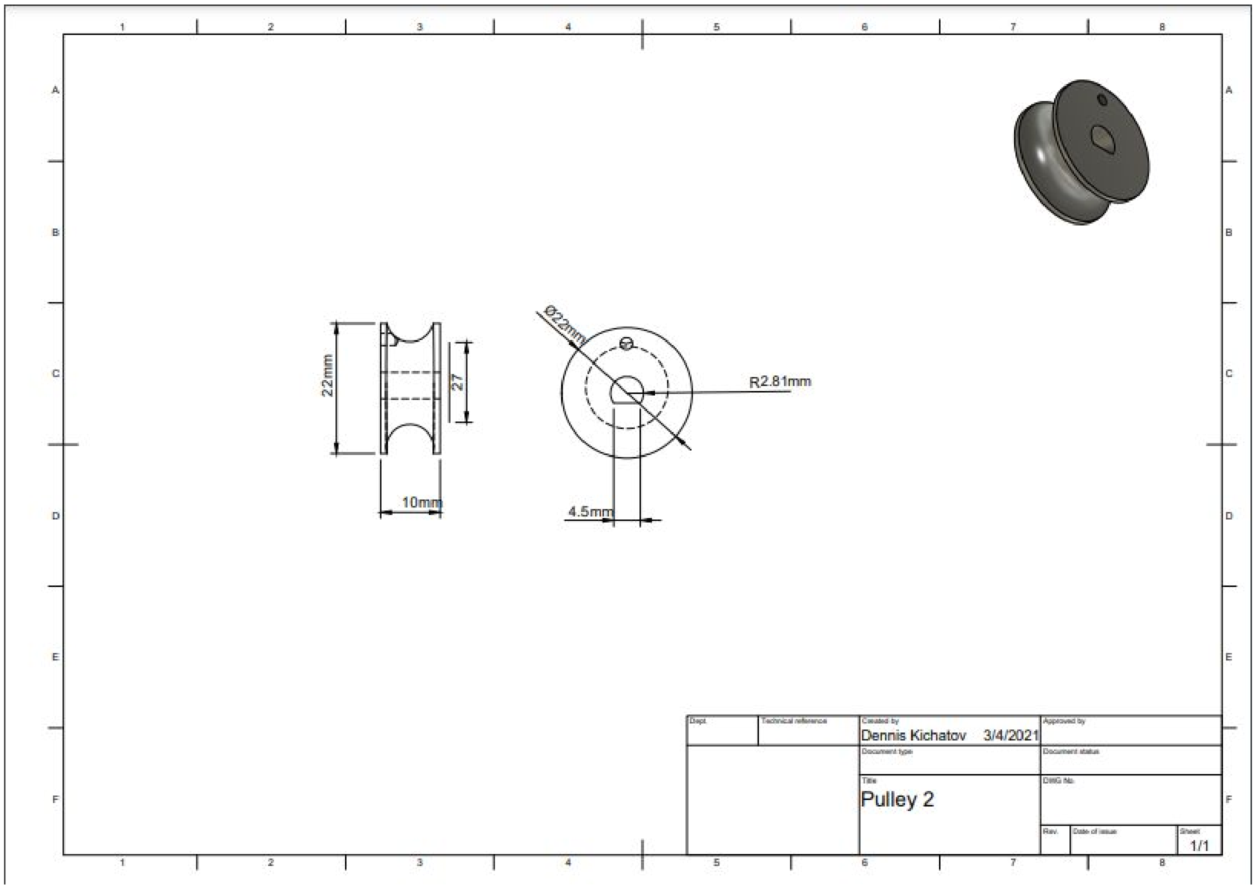

Figure 7: Pulley Model

This is the model of the pulley. The pulleys are twenty-two millimeters in diameter with an inner diameter of fourteen millimeters and a width of ten millimeters. The pulleys are designed to fit onto the team’s motors and spin with them. By wrapping the string around the pulleys, when the motors spin the mount will be able to move with the motors.

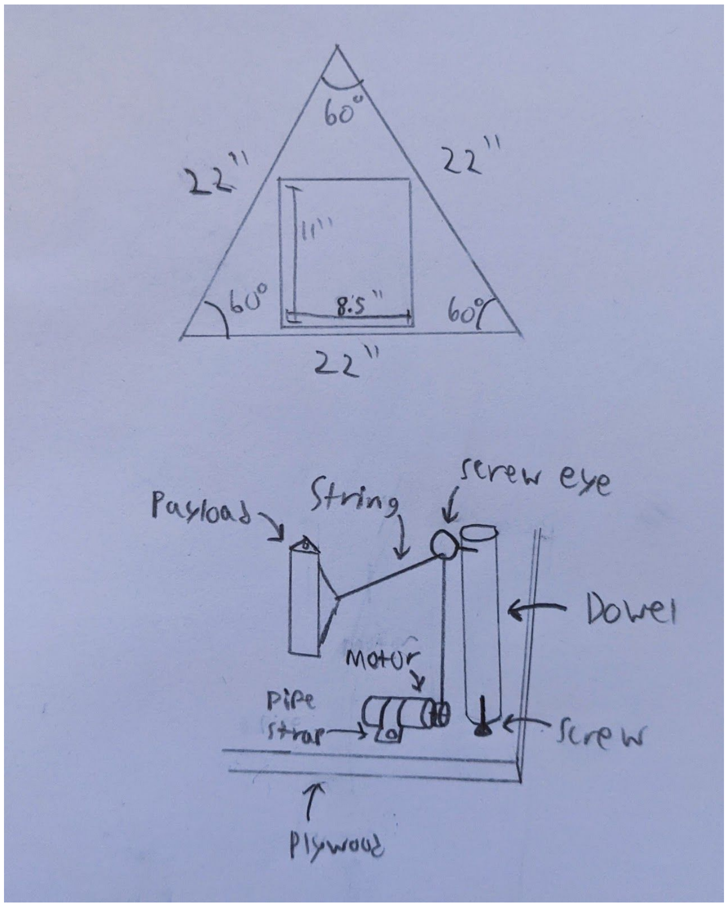

Figure 8: Payload Support System Design

This figure shows the design drawings for the payload support system, including dimensions and general design ideas. The top portion shows the dimensions of the triangle of support pylons, which is just large enough to encapsulate an 8.5 by 11 inch piece of paper. The bottom portion shows the design for each of the 3 corners of the triangle, including all the materials needed to mount the various components to the plywood board. The point of contact between the string and the screw eye is meant to be at each corner of the triangle.

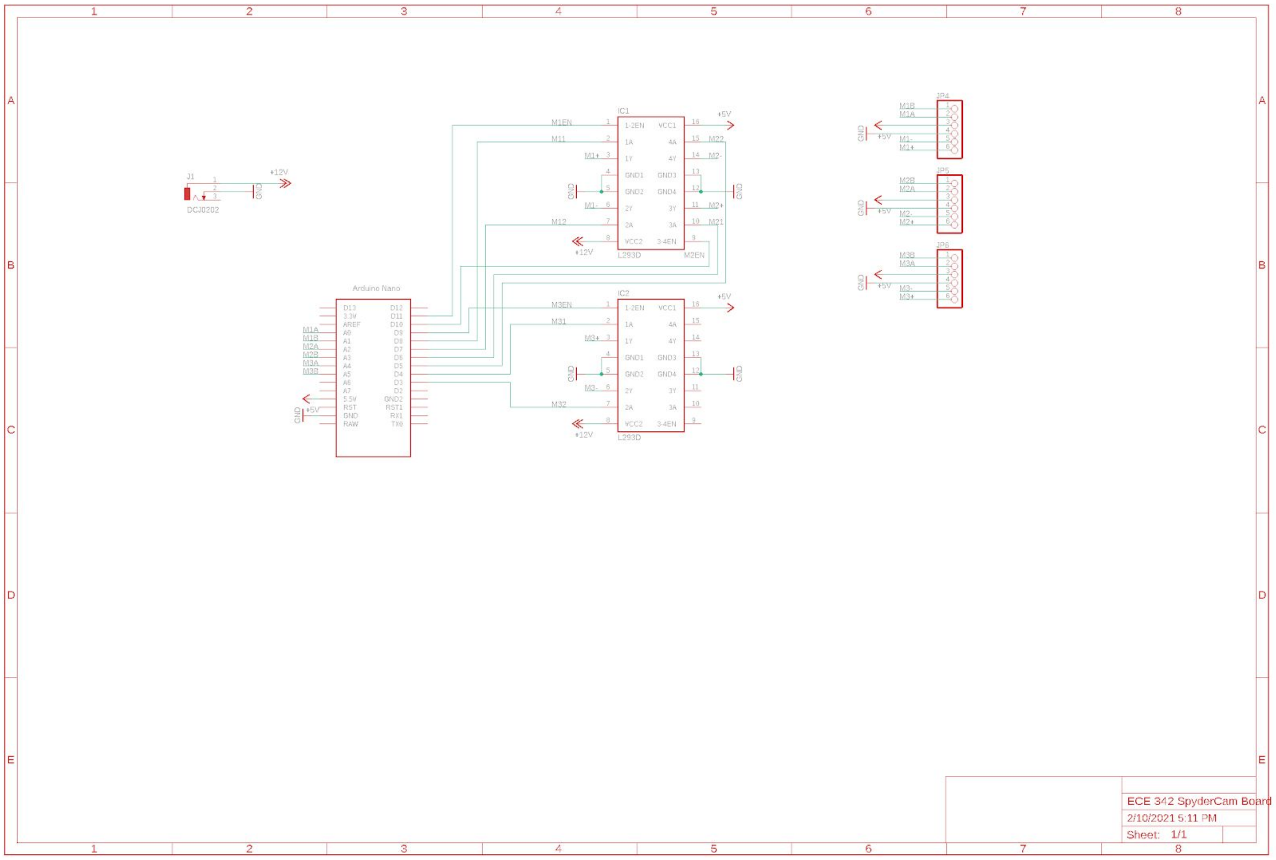

Figure 9: PCB Schematic

This PCB was designed to provide connections between the Arduino Nano Every pins, L293D motor driver chips, DC power barrel plug adapter, and pin headers for the DC motors with encoders. The PCB was designed in EAGLE, a PCB design tool that allowed for easy integration with 3D models created in Fusion360.

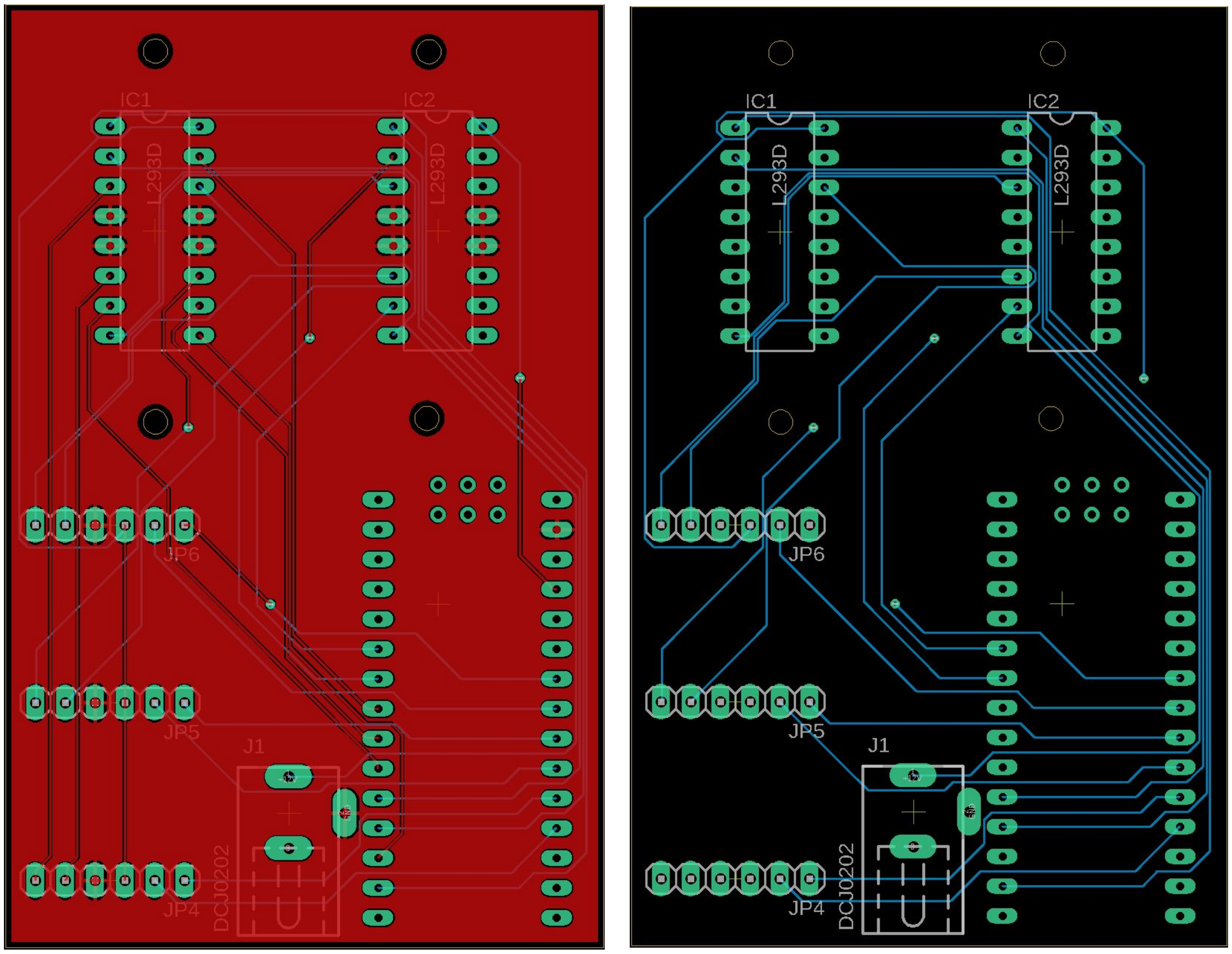

Figure 10: PCB Top and Bottom Layers

These two pictures show the top and bottom layers respectively of the PCB designed for this system. The left side is the top layer of the PCB, which is mostly filled in with a copper pour that acts as a ground plane. The majority of the wiring is on the bottom layer of the PCB, which is shown on the right side. There are also 4 holes in the top half of the PCB allowing for mounting to a 3D-printed enclosure with M2 screws.

```
//from open cv tutorial.
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <fstream>
using namespace cv;
using namespace std;
int main(int argc, char** argv)
{
      // Declare the output variables
      ofstream outfile;
      outfile.open("imgcopy.gcode");
      Mat dst, cdst, cdstP;
      string imgname;
      cout << "Enter the name of the image: ";
      cin >> imgname;
      // Loads an image
      Mat src = imread(imgname, IMREAD_GRAYSCALE);
      // Check if image is loaded fine
      if (src.empty()) {
      printf(" Error opening image\n");
      return -1;
      }
      // Edge detection
      Canny(src, dst, 50, 200, 3);
      // Copy edges to the images that will display the results in BGR
      cvtColor(dst, cdst, COLOR_GRAY2BGR);
      cdstP = cdst.clone();
      vector<Vec4i> linesP;
      HoughLinesP(dst, linesP, 1, CV_PI / 180, 50, 50, 10);
      // Draw the lines
      for (size_t i = 0; i < linesP.size(); i++)
      {
      Vec4i l = linesP[i];
      line(cdstP, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 3,
LINE_AA);
      outfile << "g00 X" << (int)l[0] << " Y" << (int)l[1] + 108 << " Z100\n";
      outfile << "g00 X" << (int)l[0] << " Y" << (int)l[1] + 108 << " Z50\n";
      outfile << "g00 X" << (int)l[2] << " Y" << (int)l[3] + 108 << " Z50\n";
      outfile << "g00 X" << (int)l[2] << " Y" << (int)l[3] + 108 << " Z100\n";
      }
      imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP);
      outfile.close();
      waitKey();
      return 0;
}
```

Figure 11: Computer Vision Code

The Purpose of this code is to take an image and create a gcode file that can be used to copy/redraw the image. It asks the user for the name or file path of the image, opens the image in grayscale, and then performs a couple of image transforms from the open cv library in order to find the lines in the image. It then takes the endpoints of the lines and outputs gcode command into a file that moves the tool above the starting point, lowers the tool down, draws the line and then raises the tool again. The specific transforms used were the Canny edge detection, and the Probabilistic Hough Line Transform. Due to how the Hough transform work, diagonal lines are not always perfectly picked up, or are represented by multiple smaller lines. Similarly, due to the canny edge detection, thicker lines might be represented by two lines. The best results come from simple images composed of thin lines. Photographs or more complex images are less likely to be correctly copied. Additionally images larger than 280x215 pixels may be cut off when converted to gcode. These issues could be fixed with more time and knowledge of computer vision.

```
#   Author: Kenton Bender
#
#   File: 342commandblock.py
#
#   Purpose:    1. Send G-code to the Arduino Nano allowing movement of motors.
#
#   Inputs:     1. Command line args for COM port.
#               2. Text file input of G-code for output.
#
#   Outputs:    1. Drawing on paper

import sys
import serial
import time


def main():
   command()
   return



def command():
   port = sys.argv[1]                  # General setup, ensure good input.
   if len(sys.argv) > 2:
       print("ERROR: Too many arguments in command line!")
       return
   if len(sys.argv) < 2:
       print("ERROR: Not enough arguments in command line!")
       return
   print('PORT =', port)               # Confirm to operator what port they are
using.
   io = serial.Serial()                # Open serial, initialize below.
Commands are separate for clarity of setup.
   io.baudrate = 9600
   io.port = port
   io.timeout = .01
   io.open()
   io.flush()
   print("MODE: Sending.")                 # Confirm to operator what mode they
are in.
   infile = open("imgcopy.gcode", "r")     # Open G-code file located in local
directory.
   lines = infile.readlines()
   # The following for loop reads from G-code and sends G-code to the Arduino
Nano over serial.
   # To prevent overflow, a line is transmitted, then a response is awaited
before sending another line.
   for x in range(0, len(lines) - 1):
       io.write(lines[x].encode())
       while io.in_waiting == 0:
           time.sleep(.05)         # Check for response every 50ms
#        setup = io.read(50)        # Proof code
#        print(setup)               # More proof code
       io.reset_input_buffer()     # Comment this line and uncomment proof code
if you want to test arduino response
   infile.close()
   io.close()
   return


main()
```

Figure 12: Computer Controller Code

The above code takes input from the .gcode file generated by the computer imaging software and sends it line by line to the arduino. It first opens serial communication, then the file, then reads the number of lines it will send. It then loops through the file reading a line, sending it over serial, and then waiting for a response to send again, polling at a rate of 20Hz.

Materials	Cost (USD)
M2 screws	$0.99
M5 Screws	$0.99
3D Prints	$16
Sandpaper	$3.17
3x Encoder Metal Gearmotor	$50.64
.4mm Kevlar kite cord	$2.00
3x Everbilt 1.5 inch screws	$0.75
3x Everbilt Stainless Steel screw eyes	$1.18
1 inch diameter by 3 feet long dowel	$5.88
2 feet by 2 feet sanded plywood board	$10.37
Arduino Nano Every	$10.90
Adafruit TTL Serial Camera	$39.95
Adafruit SD Card Breakout Board	$7.50
Micro SD Card	$9.95
Total	$160.27

Figure 13: Bill of Materials

This figure is the bill of materials for the whole team. The budget for the team was about $120. The price for this project was a bit over the team's budget. However by paying the premium the team was able to construct a working project that we are proud of.