ECE 342 Final Project

Developer Guide

Authors:
Benjamin Adams
Miles Drake
Trevor Horine
Felipe Orrico Scognamiglio

March 5, 2021
Instructor: Matthew Shuman

Contents

1 System Overview

Our team developed a SpyderCam style payload positioning system that moves a payload around in an 8.5 by 11-inch area by using three strings that connect a central payload to pylons at the corners of an equilateral triangle. An Arduino AT MEGA2560 is used to control the stepper motors that allow for the payload to change position, the Arduino also is used to measure the sensor values from the RFID and light sensors on the payload. G-Code commands generated by a MATLAB GUI are used to provide the Arduino with instructions on where to move the payload.

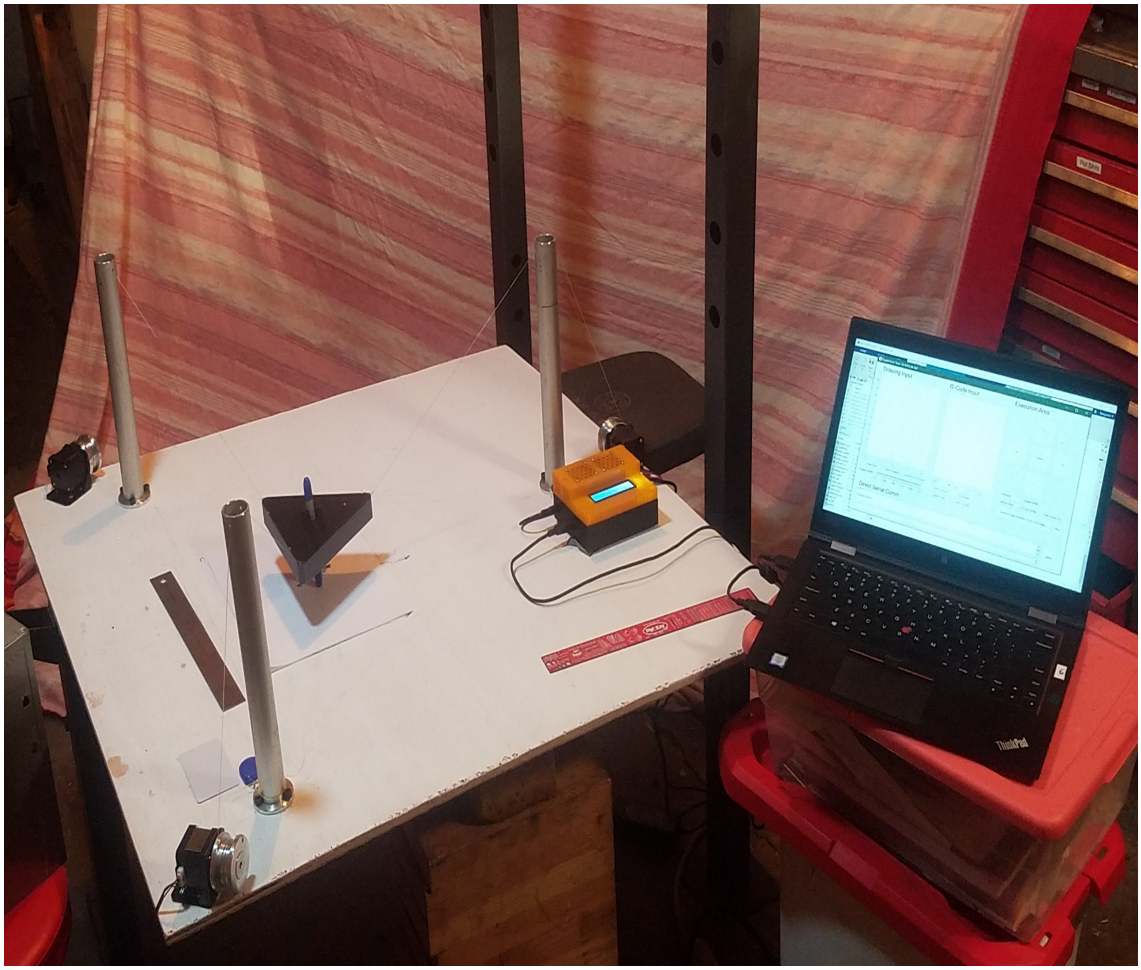

Figure 1: This is an image of what the completed project looks like.

1.1 System Block Diagram

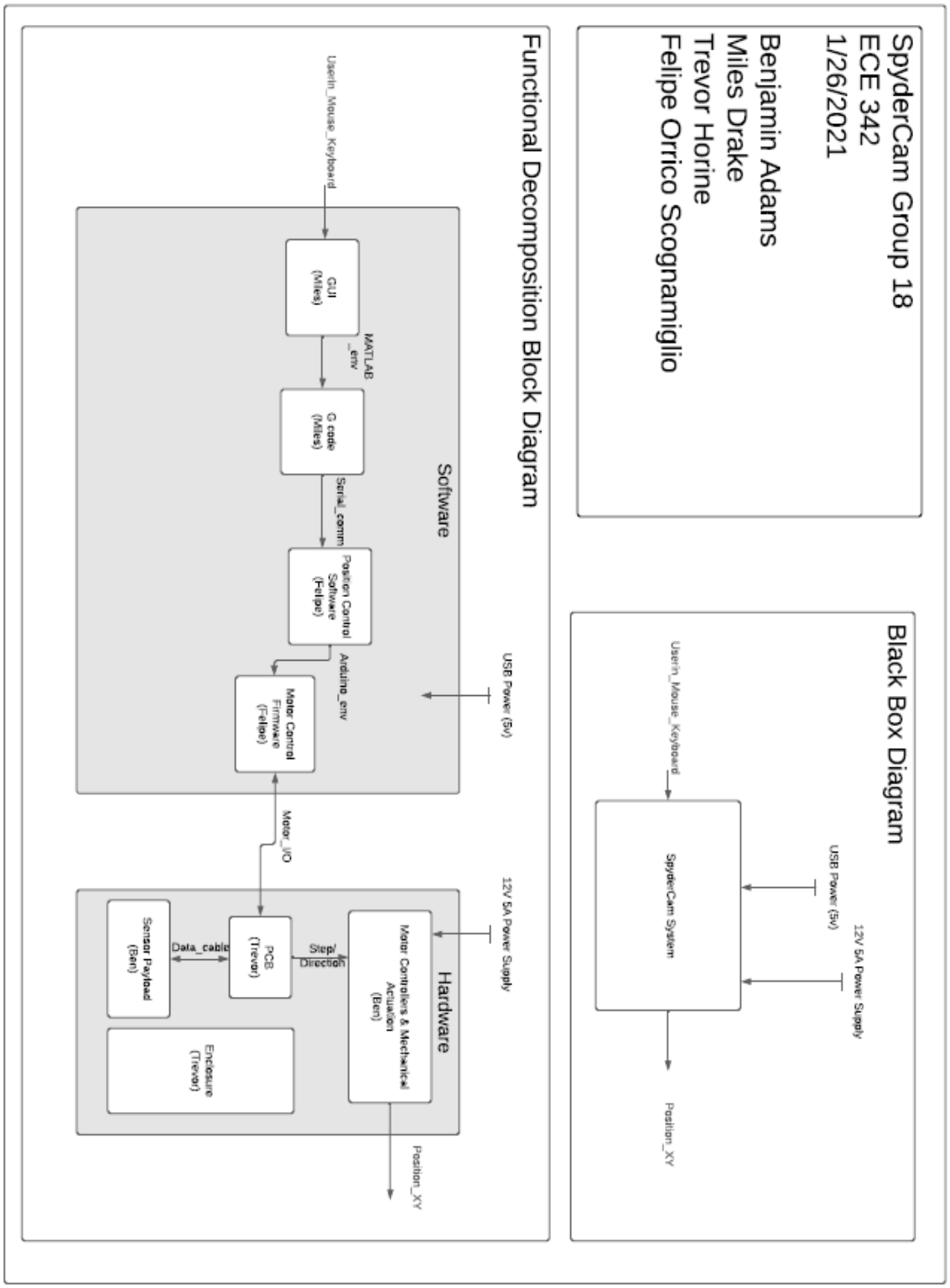

Figure 2: This is the block diagram of the whole system.

2 Electrical Specifications

The Arduino AT MEGA 2560 is powered off the USB that is also used for serial communication with the computer and MATLAB script that provides G-code commands for the system to perform. In addition, the system also has a 12 volt five amp power input that supplies the higher voltage and current that is required to drive three stepper motors used to move the payload around. With the exception of the outputs from the DRV8825 stepper motor drivers to the motors the rest of the system uses five volt logic.

2.1 System Interface Table

Interface Definitions	
Userin_Mouse_Keyboard	PS/2 Keyboard Interface Vcc = 5.0V
MATLAB _env	Internally Defined MATLAB Variables Floating-point precision
Serial_comm	Protocol: UART 115200 baud G-Code Commands
Arduino_env	Internally Defined Arduino Variables See ATMEGA 2560 DS
Motor_I/O	10 Digital Pins Vmin = 0V Vmax = 5V
Step/Direction	Vmin = -0.5 V Vmax = 7.0 V
Data_cable	6 Pin High Speed UART (115200 Baud) Vmin = 0V Vmax = 5V
USB_Power	V = 5 V I = 0.5 A P = 2.5 W
Motor_Power	V = 12.0V I = 1.5 A (nominal) I <= 1.1 A (Actual)
Position_XY	x/y payload position spanning 23.00" equilateral triangle Lateral Deviation ≤0.25" per 10" travel

Figure 3: This is the interface table for the system.

2.2 Schematics

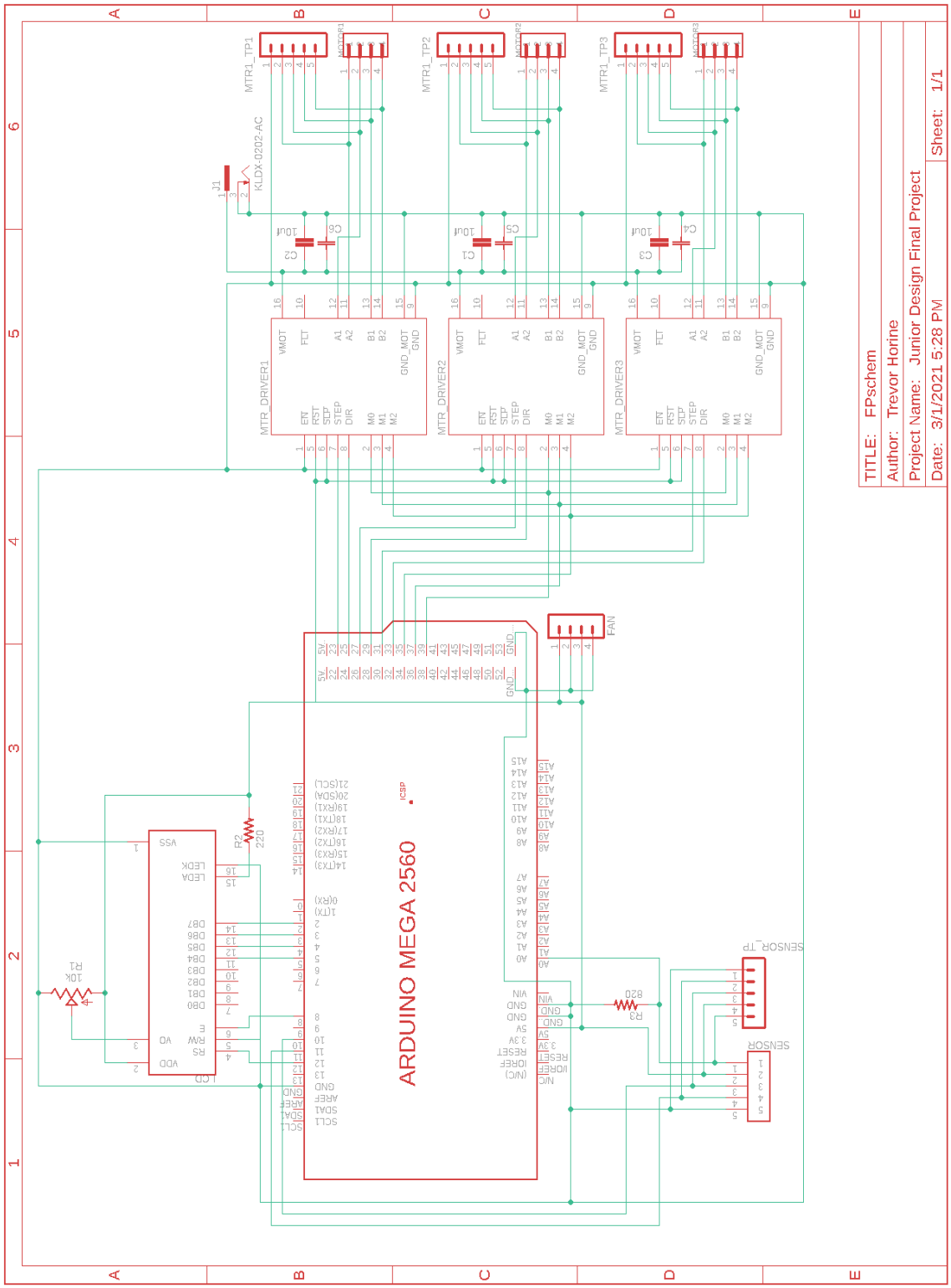

Figure 4: Detailed schematic of the system, motors connect to the connectors marked MOTOR1, MOTOR2, and MOTOR3, the sensor payload connects to the connector labeled SENSOR, and the fans connect to the pins labeled FAN.

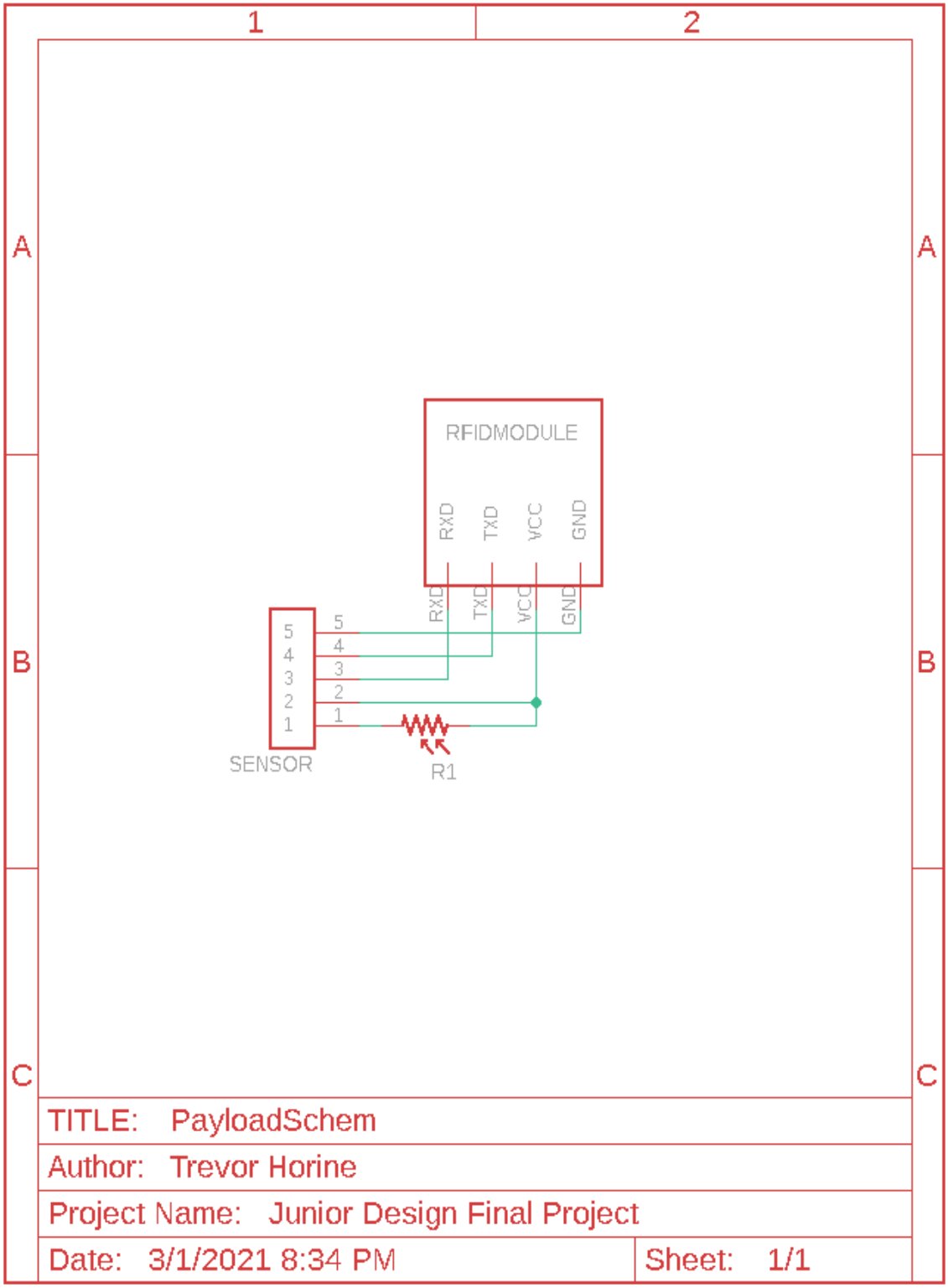

Figure 5: Detailed schematic of the payload that connects to the SENSOR connector in Figure 4.

3 User Guide

3.1 Setup

The following sections describe the initial setup and running of separate components of the Spydercam Project.

3.1.1 Hardware

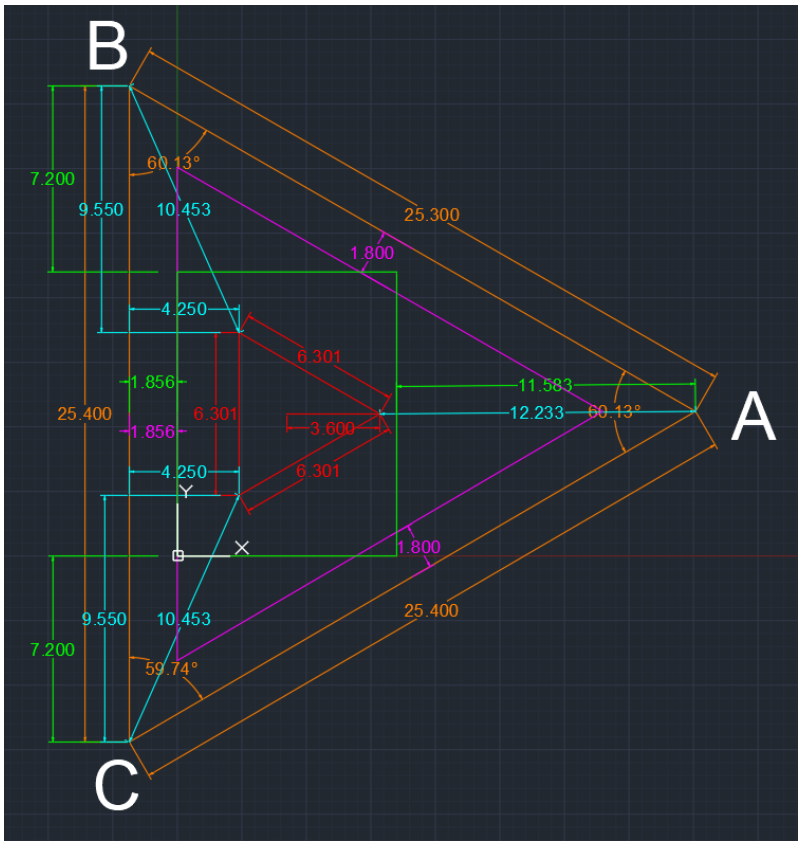

Figure 6: CAD image of relative hardware measurements with a triangular payload

Begin by setting up the pylon mounts and pylons in accordance with Figure 6 (note that different payloads can be compensated by changing software parameters (see section 3.1.2, but the pictured payload does not alter other dimensions). Dimensions pictured are from the point where the thread leaves the pylon, so some basic shop math is required. Once positioned, the pylon feet will balance the pylon, allowing for easy positioning before installation. Carefully mark all of the bolt hole positions in the pylon feet, then drill 5/16" holes to allow the insertion of $1/4 - 20$" socket-headed cap screw. Install $1/4 - 20$ screws, washers, and bolts, and tighten. Carefully position the nema 17 motor mounts outside of the triangle, allowing for a clear trajectory for the thread to travel, and bolt down after making sure that motor cables are long enough to reach the microcontroller box. After wiring up the stepper motors, the microcontroller enclosure can be mounted to to the frame. Finally, care should be taken in accurately positioning any paper inside the drawing area. Paper can be easily secured with double-sided tape, and marking the paper registration makes the reloading of paper more efficient.

The final hardware should look like Figure 7.

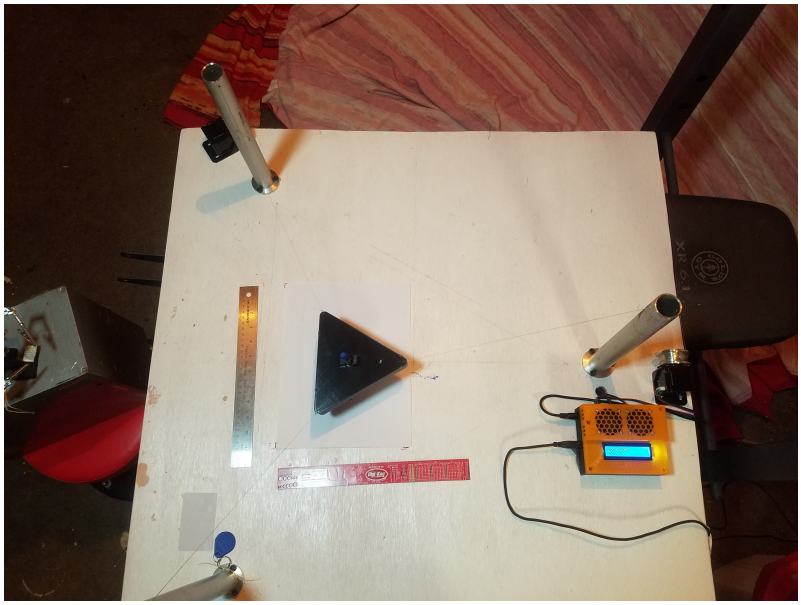

Figure 7: Final Hardware- top view

3.1.2 Software - Arduino

Before installing the Arduino software, be sure that the payload dimensions in lines `56-60` match the payload that you are using. After this is verified, load the provided Arduino code and libraries to the Arduino Mega using the Arduino IDE. After the program has loaded, the LCD display will request you to input some current values so that it knows where the payload is located, more precisely, it requests X, Y and Z (i.e. current payload X and Y location with relation to the C-edge of the paper, and the current height of the payload from paper-level to the top of the payload (where the threads are attached). After all requested values are input, the Arduino side of the setup is complete, and it is ready to receive G-Code inputs to move the payload or perform another task.

3.1.3 Software - MATLAB

To set up the MATLAB GUI for this system, load spyderGUI.m (8.2) into MATLAB. Replace the value "COM3" on line 38 to the name of the serial port being used on your computer. Plug in the Arduino to the specified serial port using a USB cable and then run the program. If the serial device is not properly connected, MATLAB will throw an error and exit the program. If this is the case, reconnect the Arduino and confirm that the proper serial port is specified. Once the main GUI window opens up, setup is complete and MATLAB is ready to send commands. Before sending any G-Code, the direct serial communication section of the GUI should be used to send the initial location values of the payload, as stated in section 3.1.2 above.

3.2 Operation

Operation of the SpyderCam is viable in two different ways. It is possible to control the hardware by inputting commands straight to the Arduino or through the MATLAB GUI.

3.2.1 MATLAB

The MATLAB portion of the system serves as a graphical user interface (GUI). It's purpose is to take instructions from the user, send those instructions to the Arduino, then receive and display sensor and location data from the Arduino. There are four different ways that a user can enter instructions into the MATLAB GUI: By typing a sequence of G-Code, by making a drawing, by pressing incremental movement buttons, or by sending serial commands directly to the Arduino. The layout of the GUI can be seen in Figure 9.

The main focus of this GUI is the drawing input area. After making a line drawing in this area, the user can convert it to G-Code which will be sent to the G-Code text input box. From here, it can be edited using the options listed below the text area, or simply edited directly by typing in the box. Once the sequence of G-Code is ready to be executed, it can be sent to the G-Code buffer (shown in Figure 10), which will send commands to the Arduino one at a time. It sends a command, then waits for a package of location and sensor information to be returned from the Arduino as discussed in section 4.1. The 'G' at the end of the information package lets the buffer know that it's time to send another command. The information received is parsed and plotted in the sensor data window, shown in Figure 11.

In addition to the main program flow described in the previous paragraph, the direct serial communication area can be used to send commands to the Arduino regardless of whether a 'G' has been sent back. This is useful for sending M6 or M2 commands, which need to interrupt a current process. Received serial data from the Arduino will also appear in this section of the GUI.

3.2.2 Arduino

Operating SpyderCam with the Arduino is quite simple. After the setup is complete, all that needs to be done is to send commands through Serial to the Arduino. Those commands are in standard G-Code format and you can find a full list of them in Figure 8.

4 Design Artifacts

4.1 Arduino Code

The Arduino code Block contains two sub-blocks: Motor Control Firmware (MCF) and Control Software (CS). Those two sub-blocks work together to translate G-Code input into correct payload movement. The CS receives the G-Code input from Serial and collects all necessary values from it. It then interprets that information and sends a move request to the MCF. The MCF upon receiving a move request will calculate the target thread lengths and the amount of steps that each motor has to perform in order to achieve the correct displacement. It will then communicate with the stepper library AccelStepper to calculate the necessary speeds for each of the motors so that they reach each of their individual displacements at the same time. The CS is also responsible to reporting back values to MATLAB such as sensor data and current thread lengths and payload position.

G/M-CODE	VAR 1	VAR 2	Var 3	VAR 4	Definition
G0	X	Y	Z	-	Takes Payload to X, Y at max speed
G1	X	Y	Z	F	Takes Payload to X, Y at F speed
G20	-	-	-	-	Mode: Inputs in Inches
G21	-	-	-	-	Mode: Inputs in Millimeters
G90	-	-	-	-	Mode: Absolute Coordinates
G91	-	-	-	-	Mode: Incremental Coordinates
M2	-	-	-	-	End Program
M6	-	-	-	-	Tool Change
C1	-	-	-	-	Send Payload location to Serial
C2	-	-	-	-	Send thread lengths to Serial

Figure 8: Table of available G-Code commands within the Control Software

After any command from Figure 8 is sent to the Arduino and executed, the Control Software will send back information to MATLAB in the following format:

X<x-location>**Y**<y-location>**Z**<z-location>**A**<thread-len-a>**B**<thread-len-b>**C**<thread-len-c>**L**<light-level>**R**<rfid-data>**G**

The letter 'G' that is sent after the information indicates that the software is ready to receive another command. When a C1 or C2 command is executed, the format is different in the way that the information is sent to MATLAB. It will include a '#' to indicate the end of the transmission cycle of the information requested.

4.2 MATLAB Code

The Drawing Input section of the MATLAB GUI allows the user to control the SpyderCam by entering a line drawing. MATLAB converts this drawing into G-Code by creating commands which move to the start of each line, lower the pen onto the paper, move along the path created by the line, then raise the pen and continue to the next line. The G-Code is all written in standard G-Code notation.

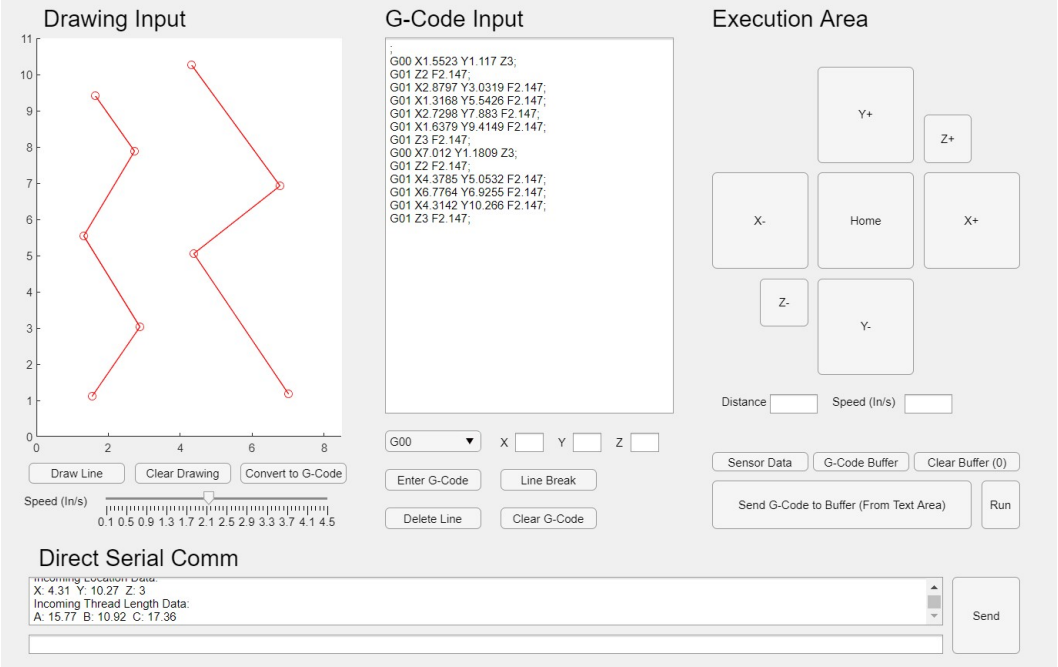

Figure 9: MATLAB GUI. Sections from left to right are: Drawing input area, G-Code text box, Execution area. The direct serial communication area is at the bottom.

G-Code which is sent to the buffer will appear in the G-Code buffer window. G-Code is sent from the buffer to the Arduino and subsequently removed from the buffer upon receiving a 'G' character through serial. The G-Code buffer window is shown in the figure below.

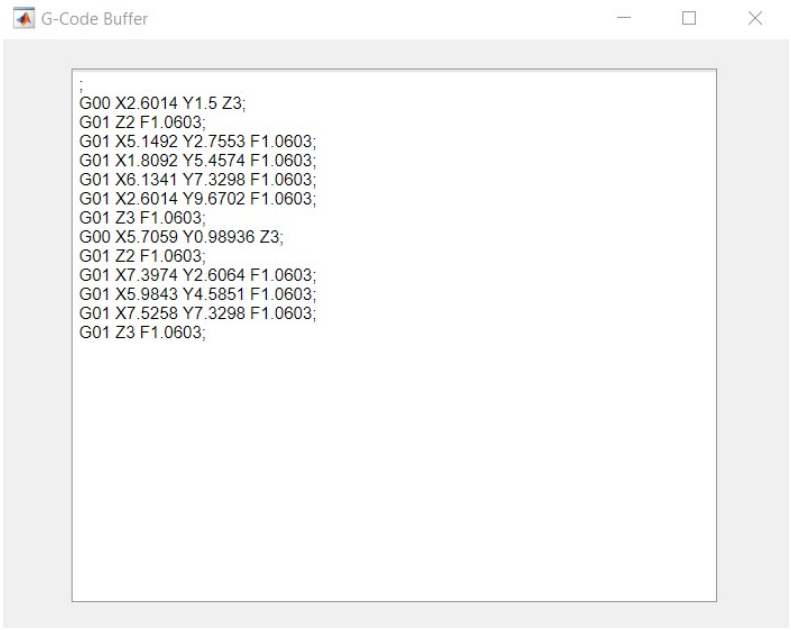

Figure 10: MATLAB GUI G-Code Buffer. This window shows a list of G-Code which is queued to be sent to the Arduino.

When the Arduino reaches its current destination, it sends back a packet of location and sensor information as described in the previous section. The information is then parsed and displayed in the Sensor Data Window, shown in the figure below.

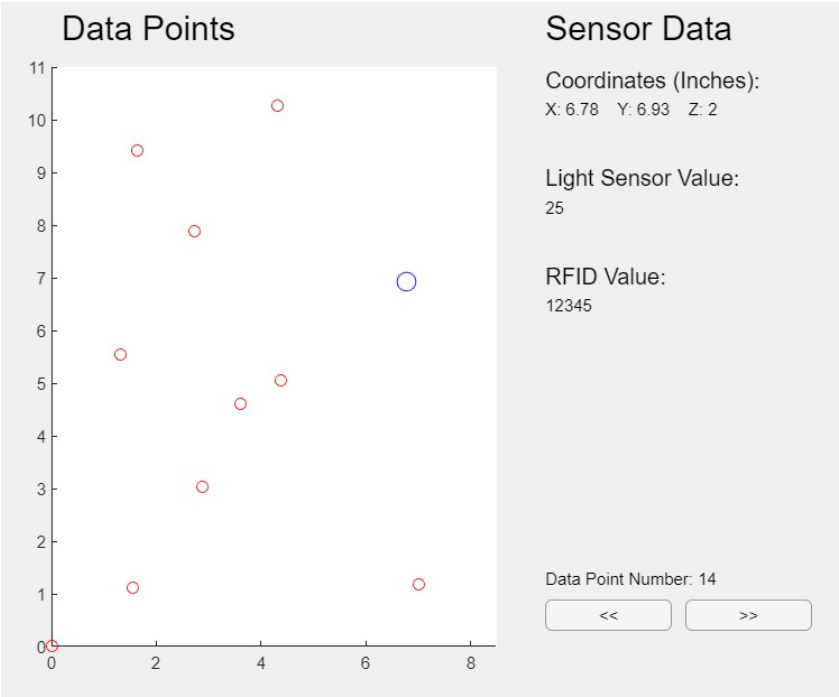

Figure 11: MATLAB GUI sensor data view. Each data point is contains a packet of sensor and location data received from the Arduino

4.3 Mechanical Subsystem

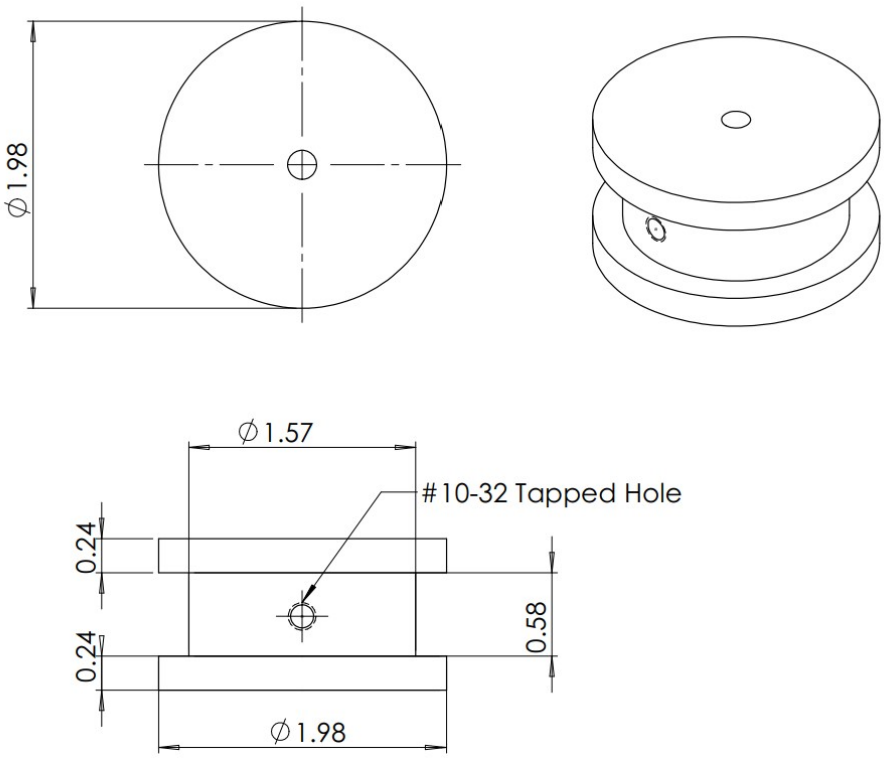

Figure 12: Aluminum motor- mounted spool blueprint

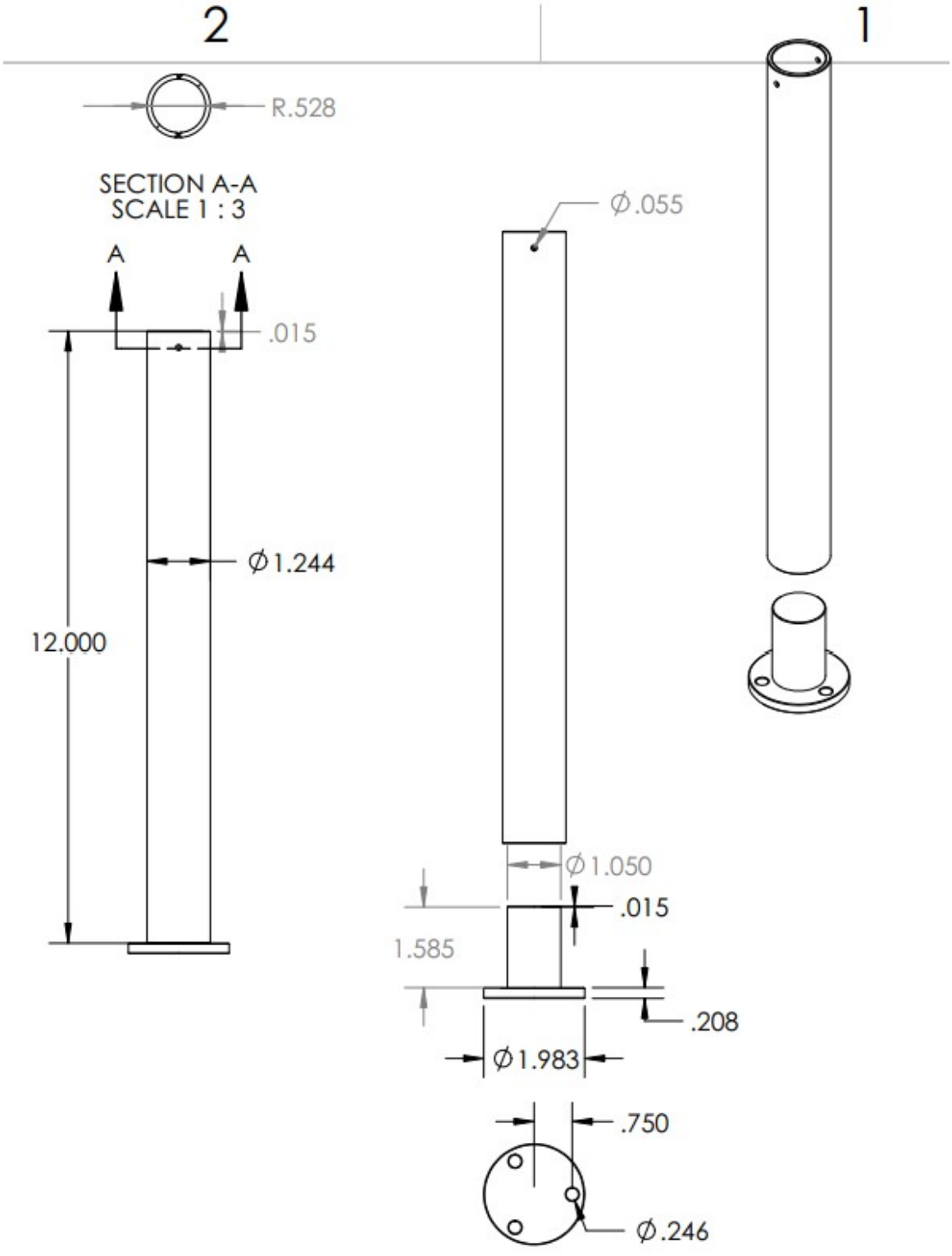

Figure 13: Aluminum pylon assembly blueprint

4.4 Payload

4.4.1 Triangular Payload

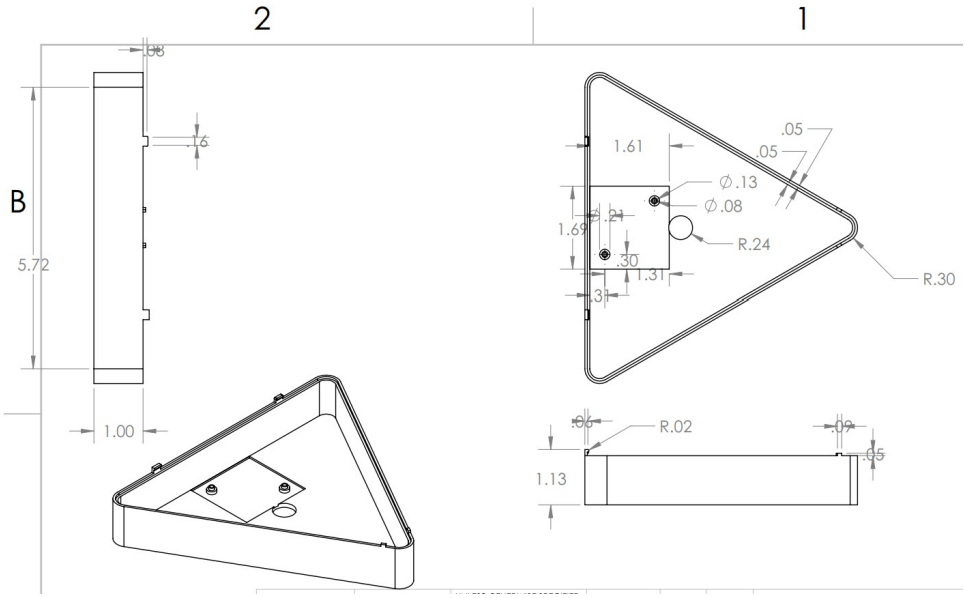

Figure 14: Triangular payload bottom part

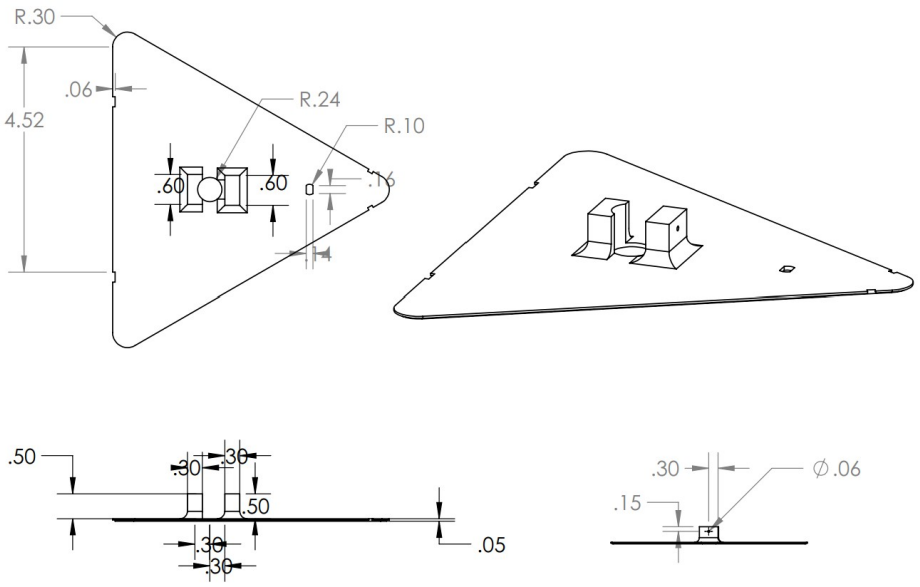

Figure 15: Triangular payload top part

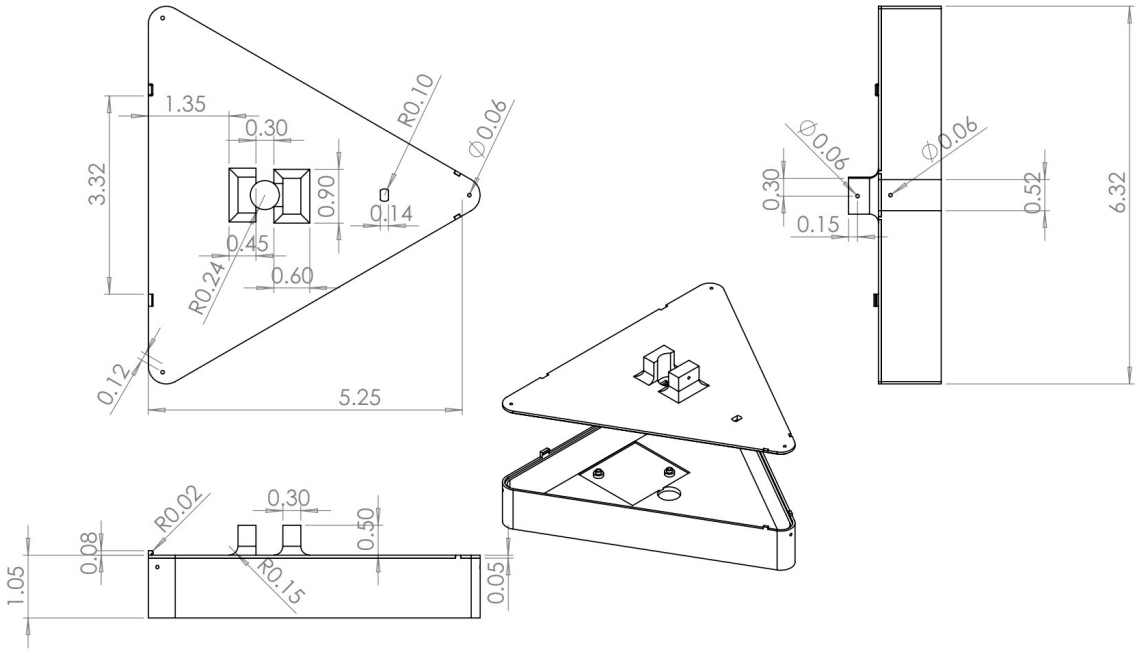

Figure 16: Triangular payload assembly- exploded view

4.4.2 Rectangular Payload

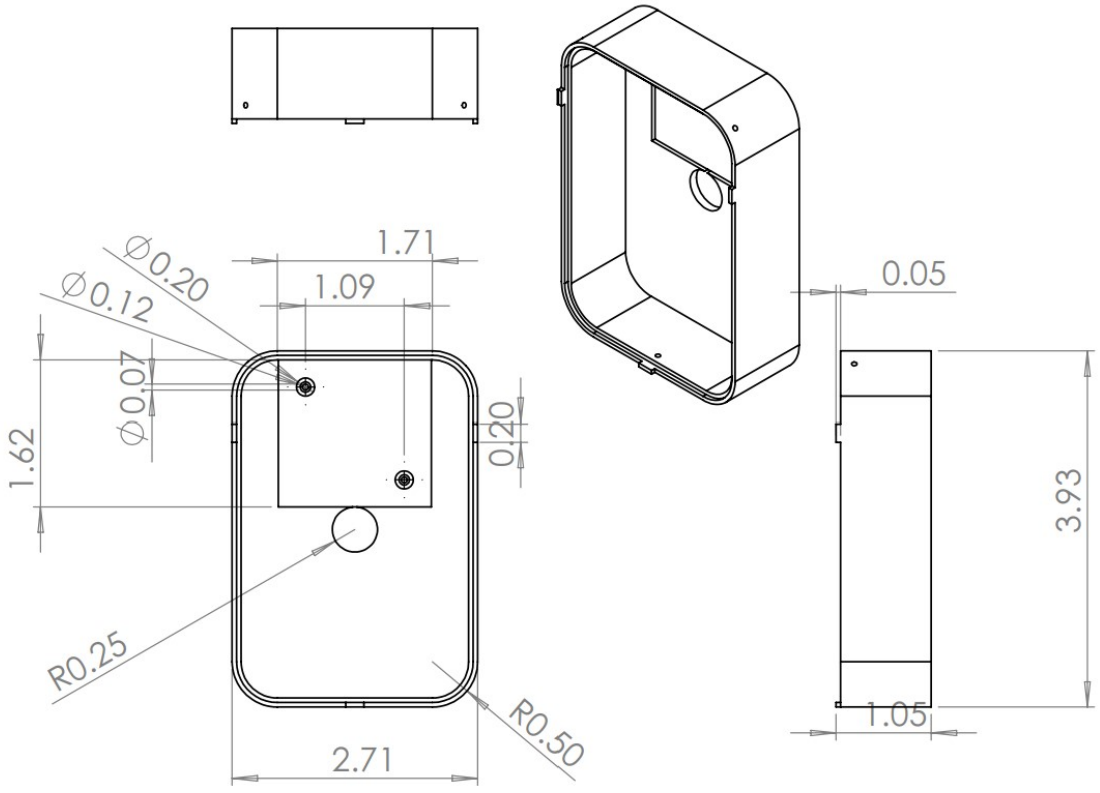

Figure 17: Rectangular payload- bottom part

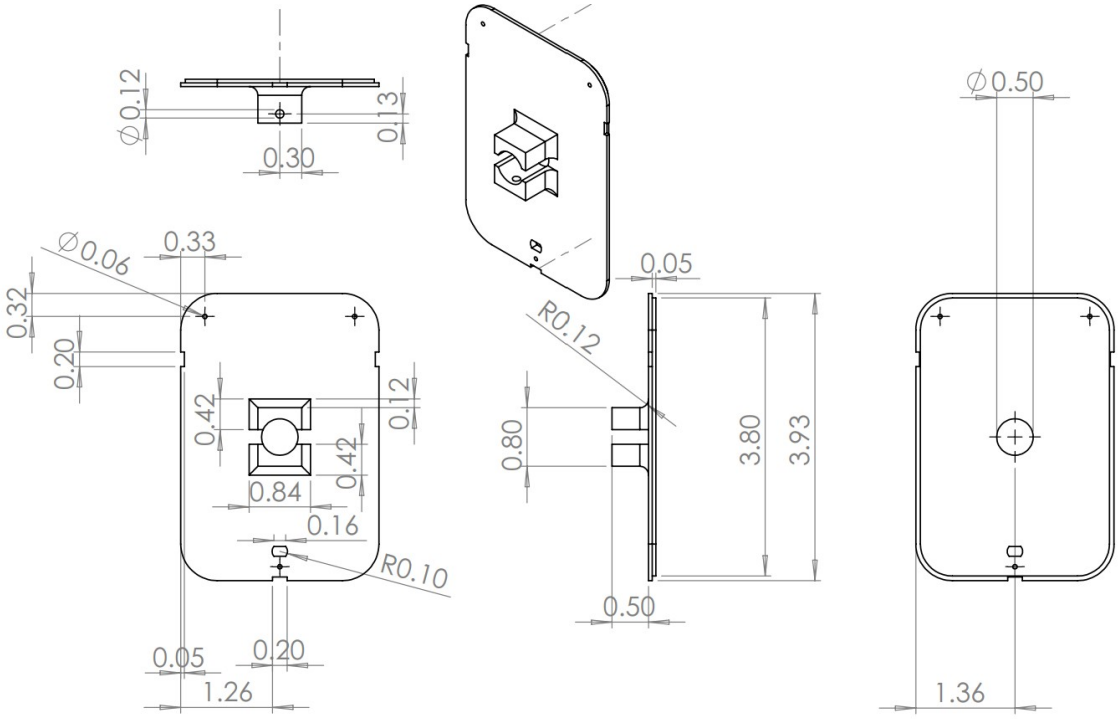

Figure 18: Rectangular payload- top part

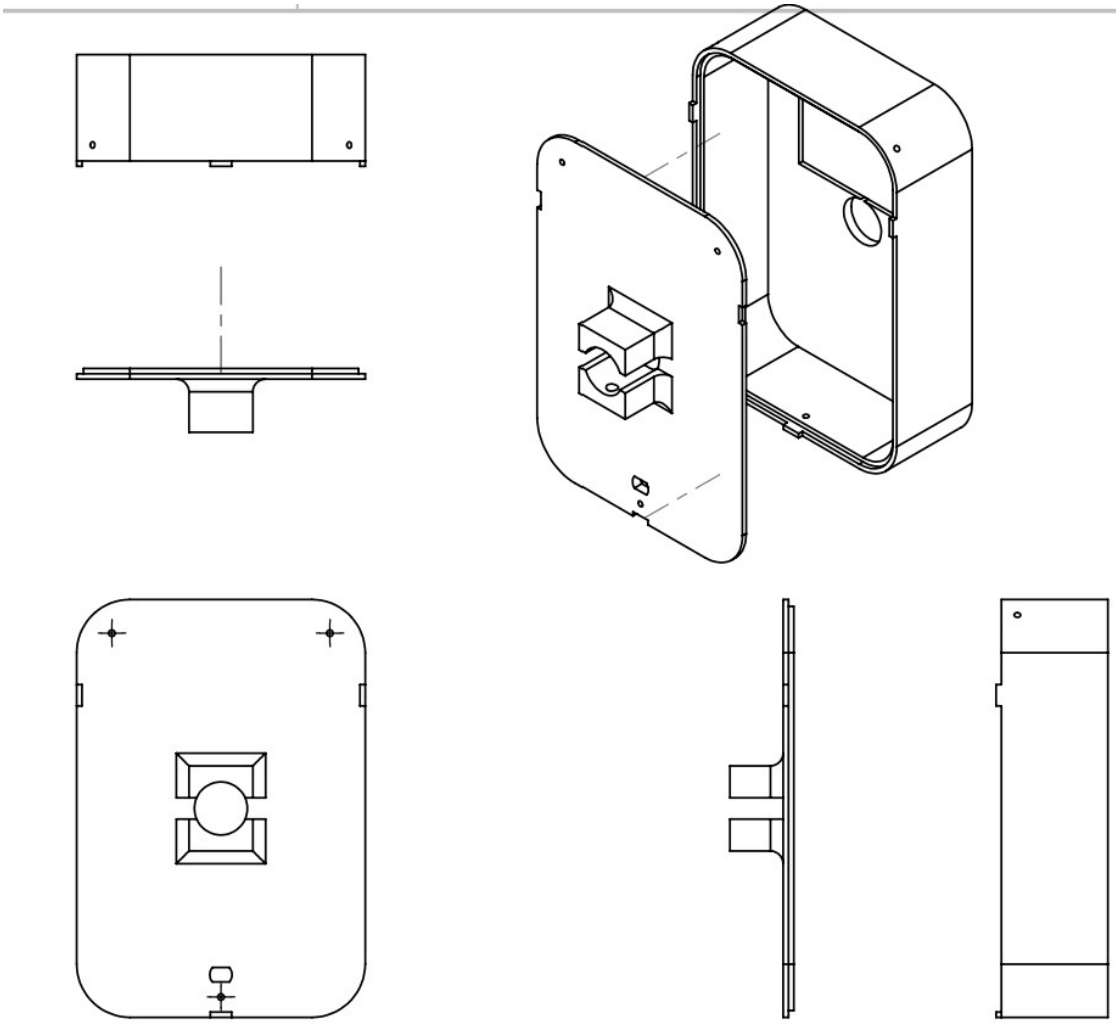

Figure 19: Rectangular payload- exploded assembly view

4.5 PCB Enclosure

The PCB enclosure was modeled using the 3D model of the PCB, such that the board would sit on supports around the mounting holes, and the top would have have supports that come down to hold the board in place. four 35 mm long M3 screws hold the enclosure together. The two 40mm by 40mm by 10mm fans also mount to the top of the enclosure to provide air circulation in the top on to the motor drivers and out the side vents. there is a window in the top to make the LCD screen visible and ports in the sides for the Arduino USB, 12 volt motor power, three motor cables and the one sensor cable. Figure 20 shows the 3D model of both haves together with the model of the PCB inside of the enclosure. Mechanical drawings for bottom and top halves of the enclosure can be found in Figures 21 and 23 respectively, followed by the models of each half in Figures 22 and 24 respectively. The final enclosure with the built PCB inside can be found in Figure 25.

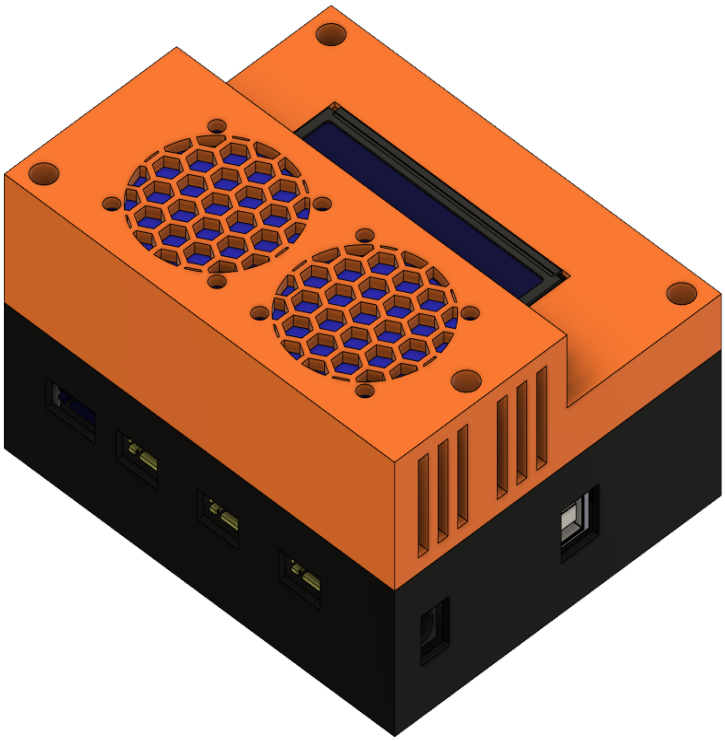

Figure 20: This is a rendering of the 3D model of what both pieces would look like together. This model has the model for the PCB inside, and has blue boxes for fans.

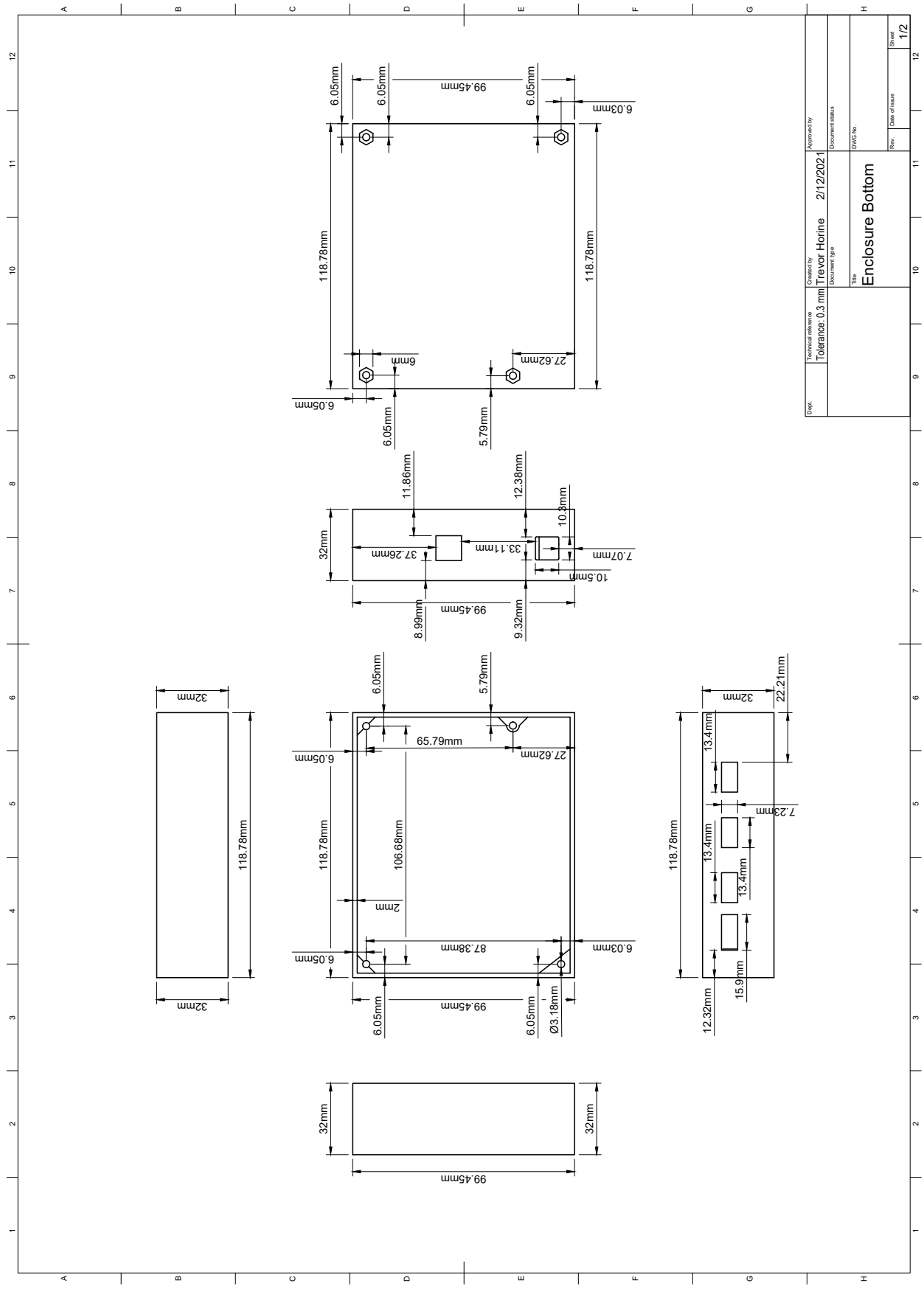

Figure 21: Mechanical drawing of the bottom half of the enclosure designed to hold the PCB.

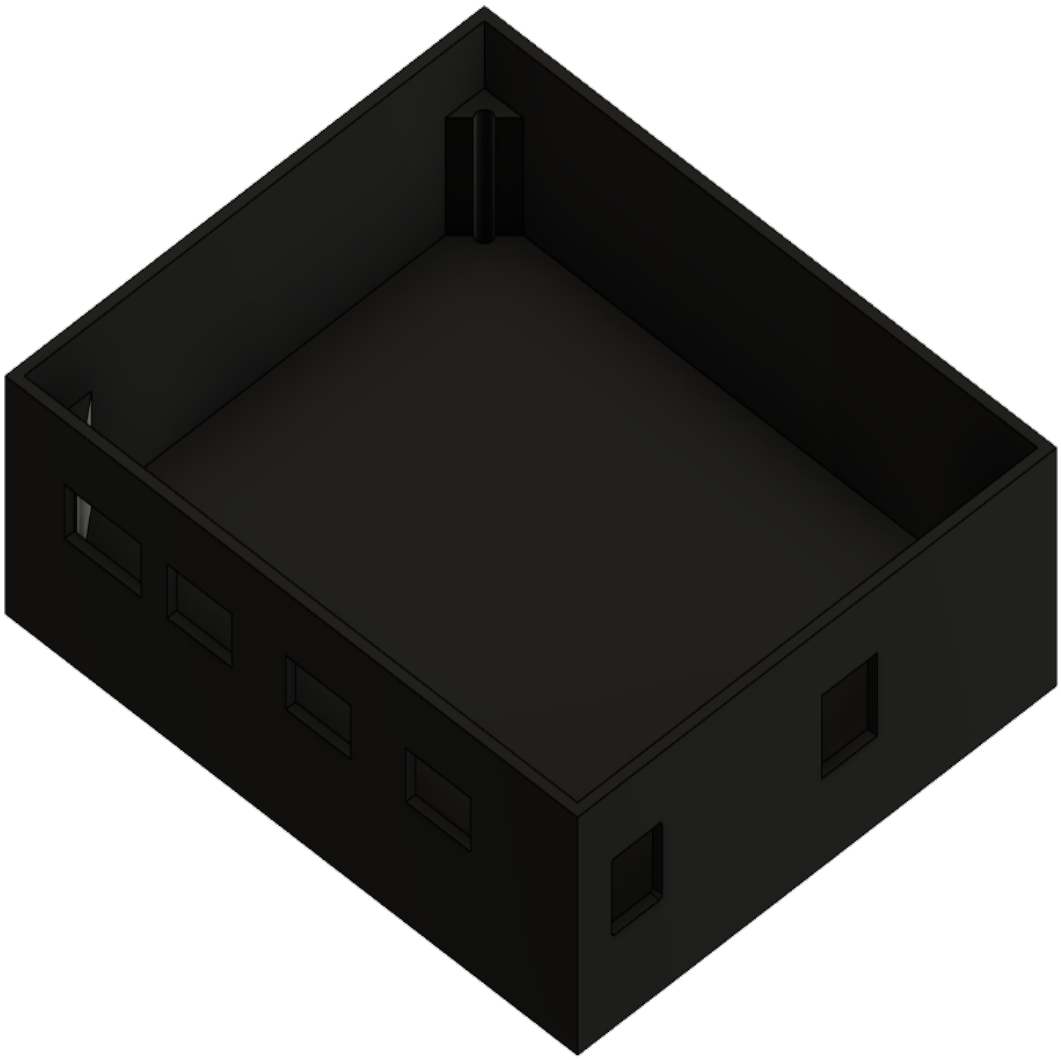

Figure 22: This is a rendering of the 3D model of the bottom half of the enclosure.

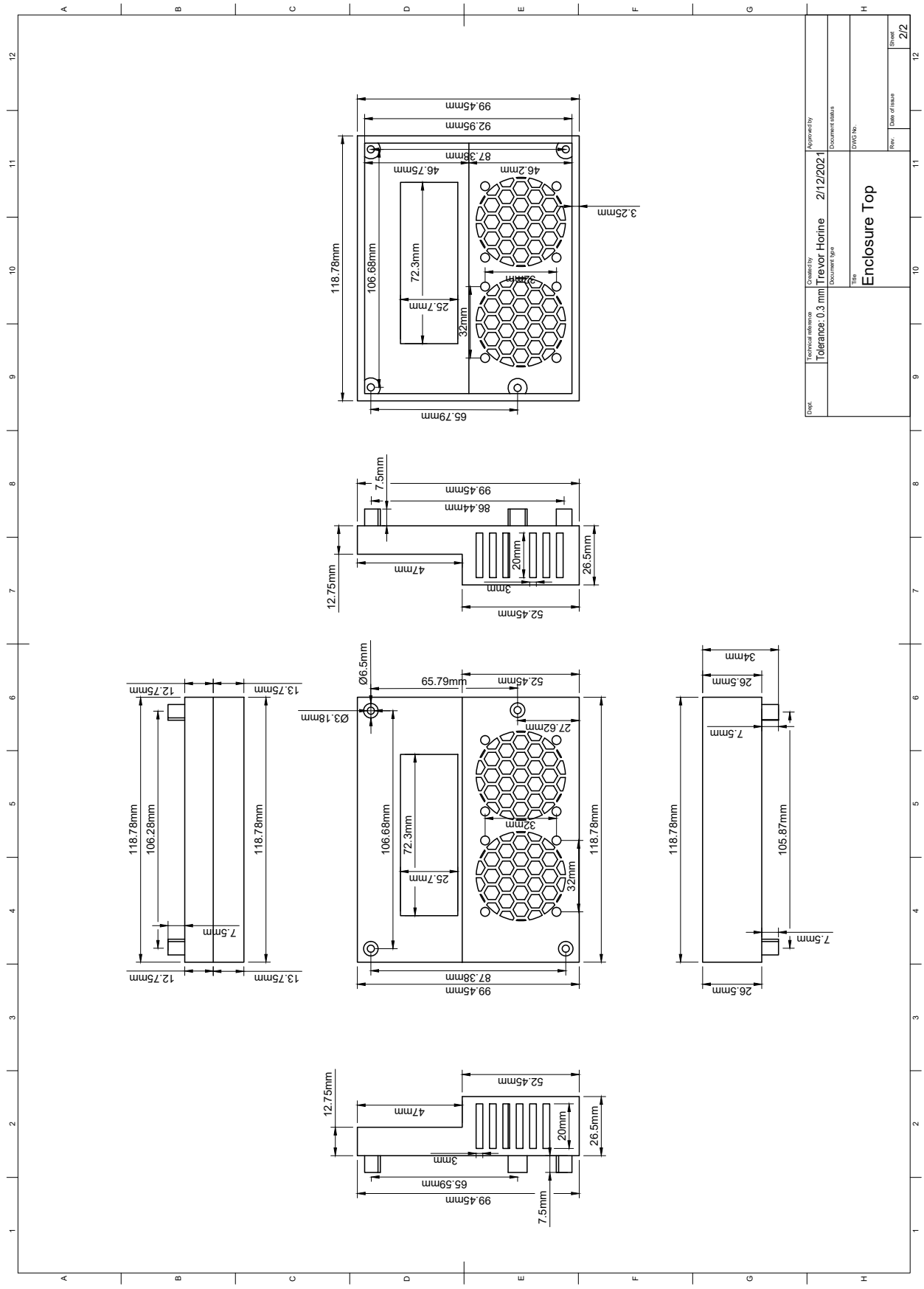

Figure 23: Mechanical drawing of the top half of the enclosure designed to hold the PCB.

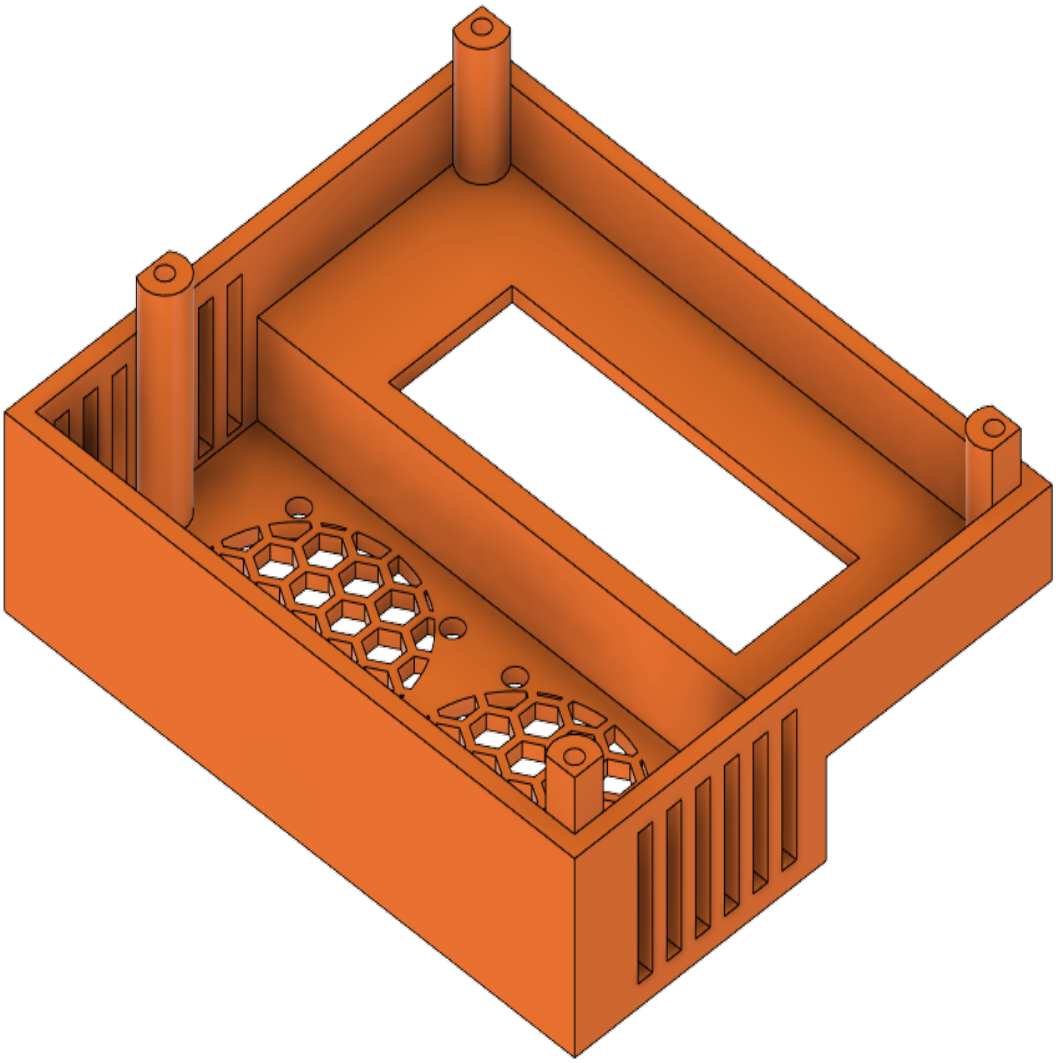

Figure 24: This is a rendering of the 3D model of the top half of the enclosure.

Figure 25: This is an image of the final enclosure with the built PCB inside.

5 PCB Information

5.1 PCB Block Diagram

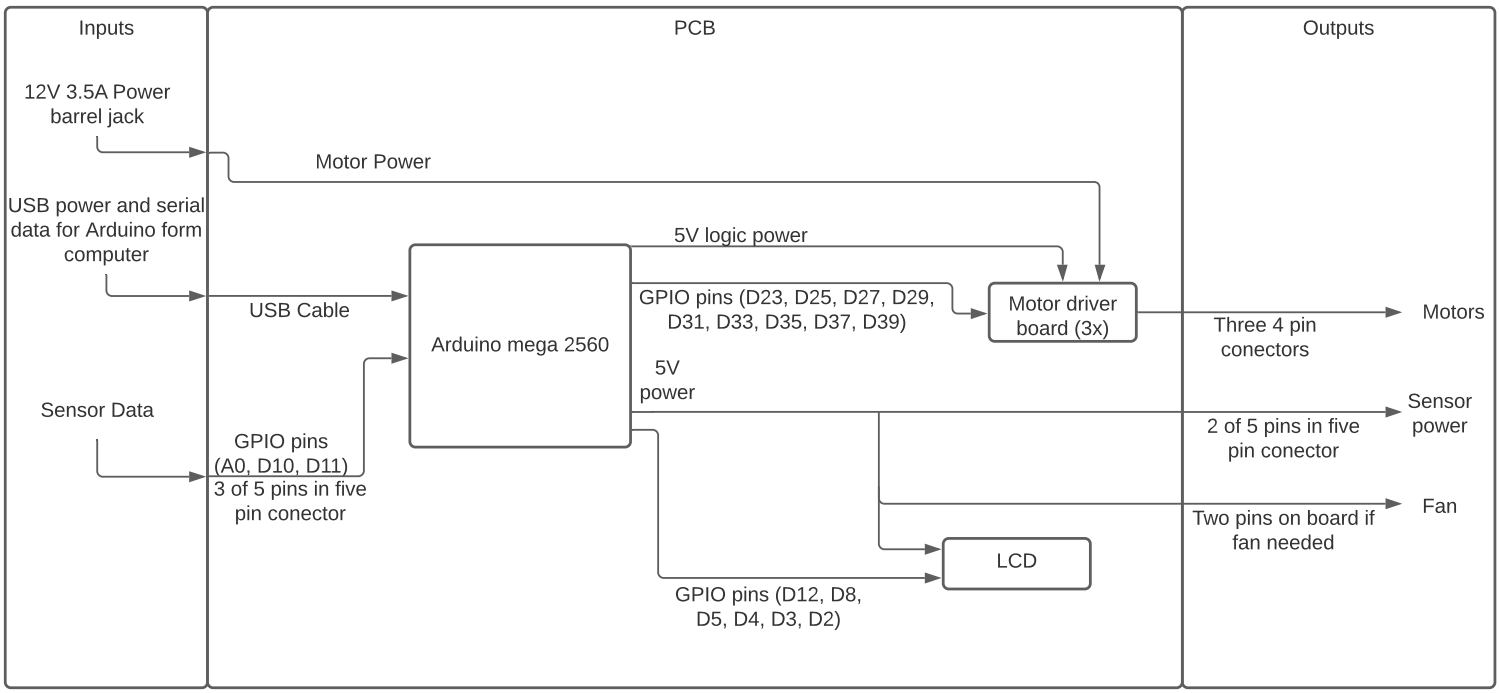

Figure 26: Block Diagram of the PCB with inputs in the inputs box on the left, outputs in the output box on the right and the blocks that make up the PCB in the middle section labeled PCB.

5.2 PCB Schematics

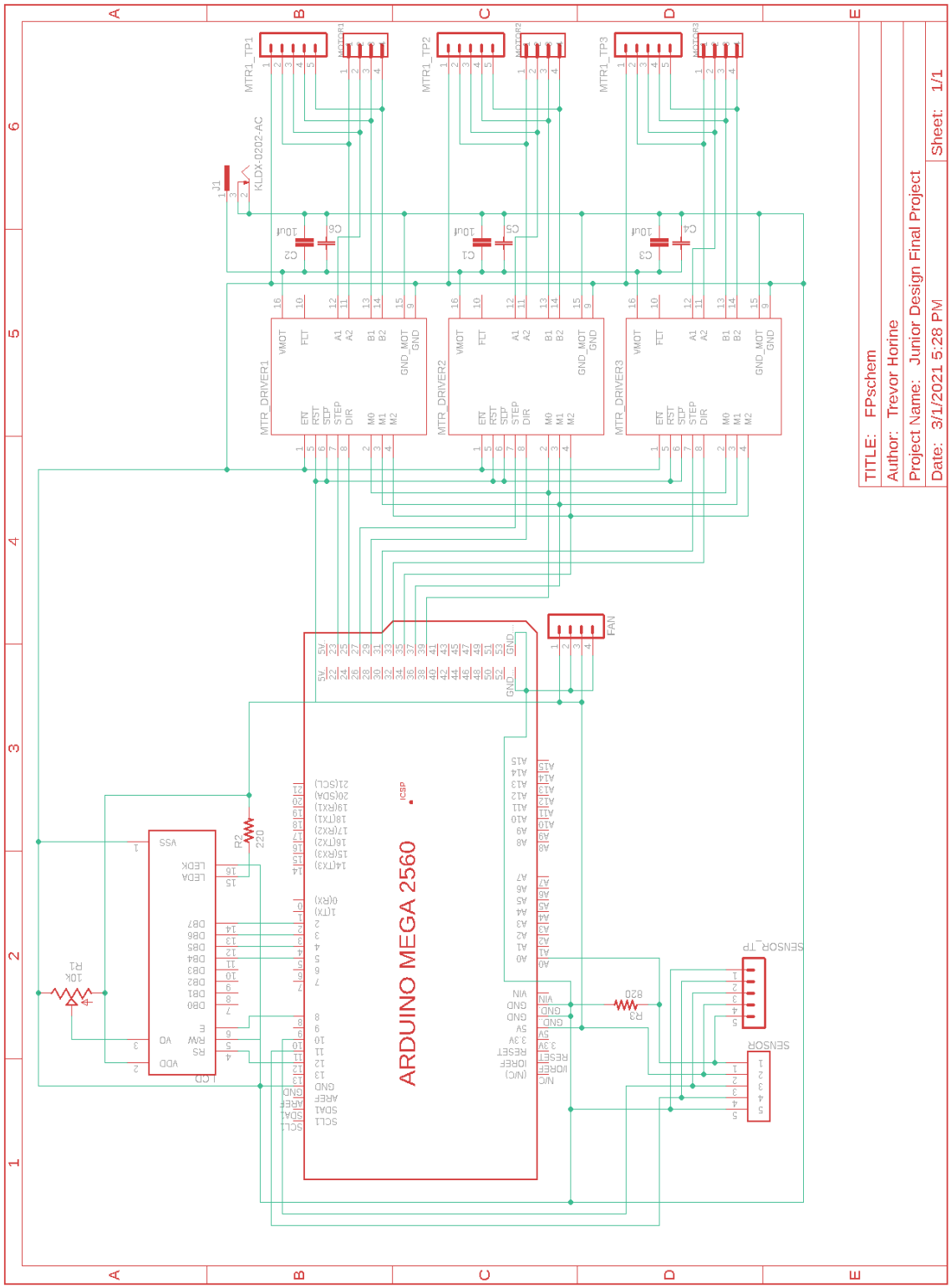

Figure 27: Detailed schematic control board, used for design layout of the PCB, which is designed as a breakout for an Arduino MEGA 2560 R3.

5.3 PCB Interface Table

Interface table	
Interface	Properties
Inputs	
USB	Supplies 5V power to Arduino. Also used a serial comunication between computer and Arduino
12V Power	12V, 3.5 A power supplied to board via a barrel jack for the motor drivers.
Sensor Inputs	Three of the five pins in the 5 pin sensor conector. The light sensor goes to GPIO pin A0, and RFID goes to GPIO pins D10 and D11.
Outputs	
Motors	Three 4 pin conectors that conect the motors to the PCB. Each of these conectors is attached to a different motor controler
Fan Power	These are a few pins on the board that are connected to 5V and GND so a fan could be added to help with cooling if needed.
Sensor Power	The remaing two pins in the 5 pin sensor conector to carry 5V and ground to the sensors on the payload.
LCD	16 pin 2x16LCD attaced to GPIO pins D12, D8, D5, D4, D3, D2, 5V and GND. This will be used for debugging. (displaying corrdinates, or sensor values.)

Figure 28: list of input and outputs, as well as some of their properties or what they will be used for.

5.4 Eagle Layout and Board Dimensions

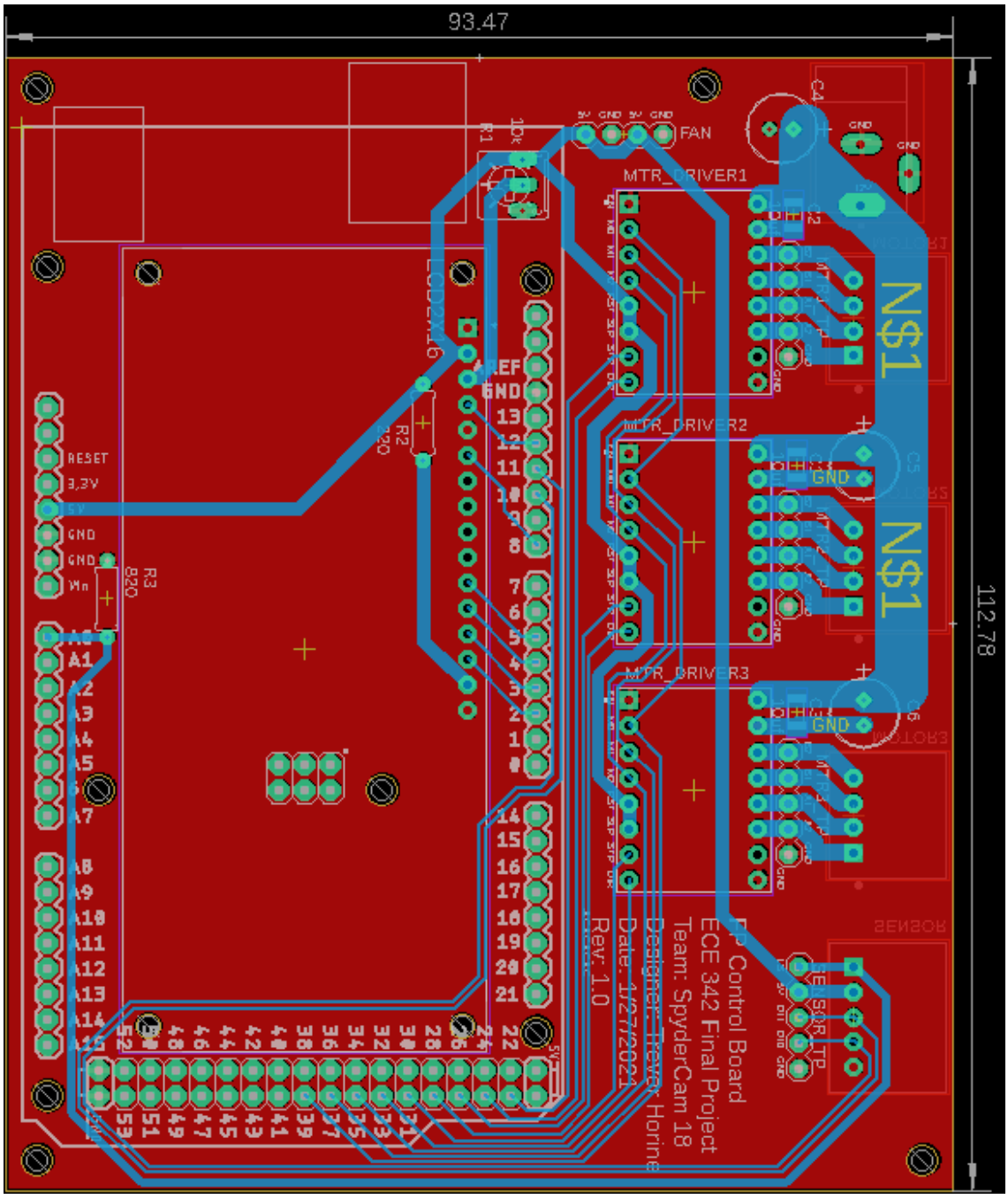

Figure 29: Layout of the PCB board built form the schematic in Figure 27, red is the bottom layer and a ground plain, blue is the top layer with traces for signals.

5.5 PCB Mechanical Drawing

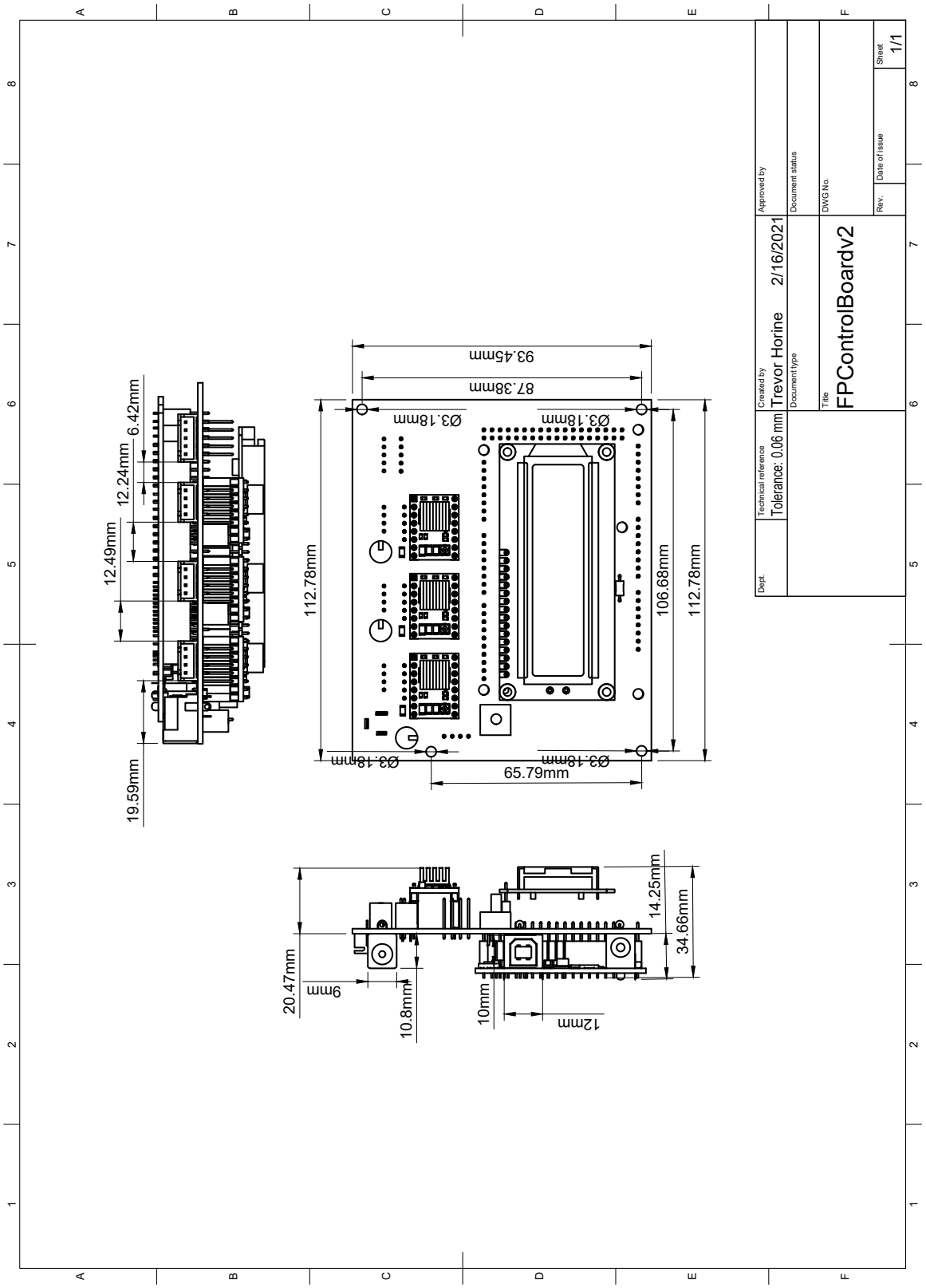

Figure 30: Mechanical drawing of PCB, with dimensions and locations of mounting holes

5.6 Board Profile

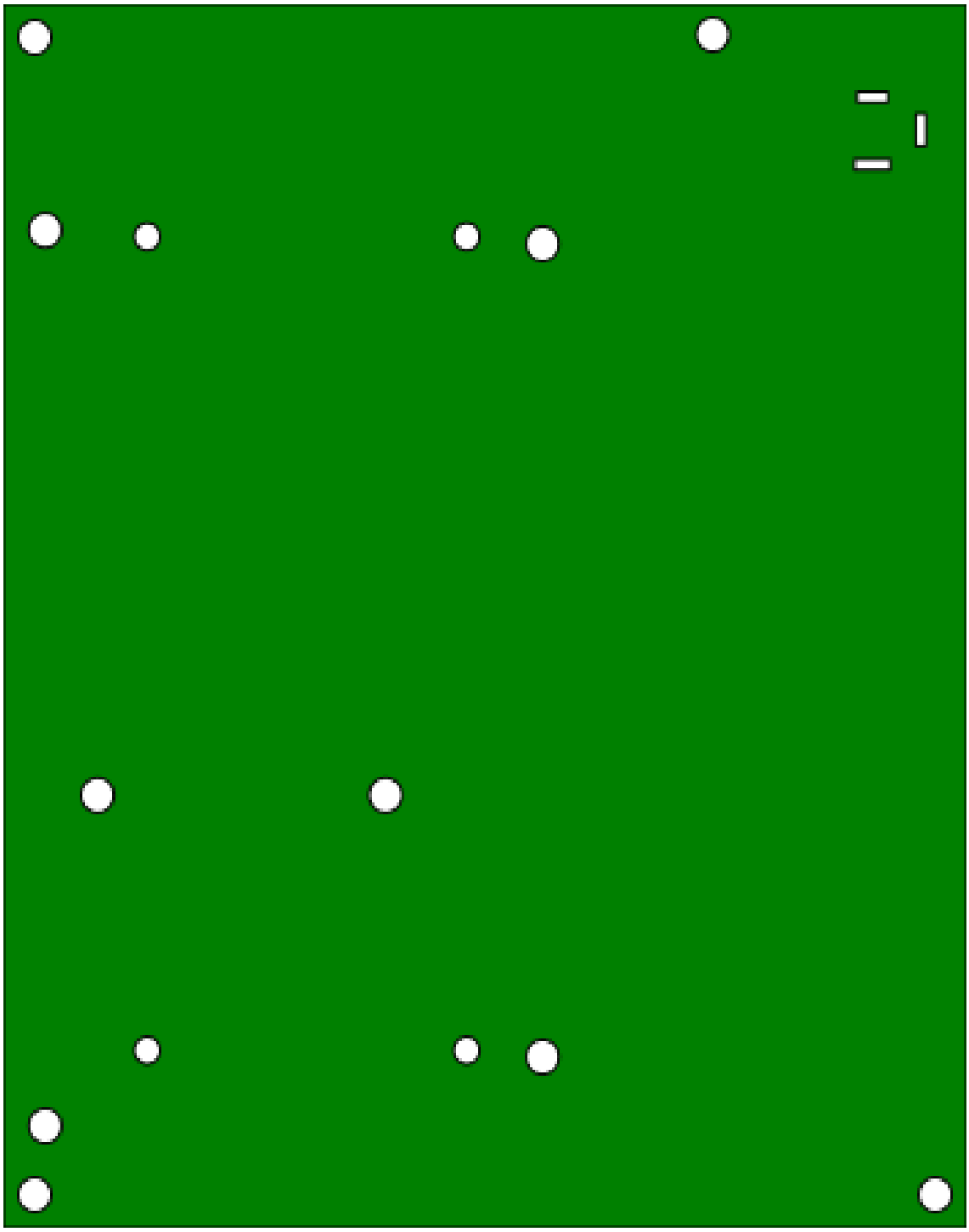

Figure 31: This is a rendering of the board profile. This image was generated from the Gerber files.

5.7 Top Silkscreen

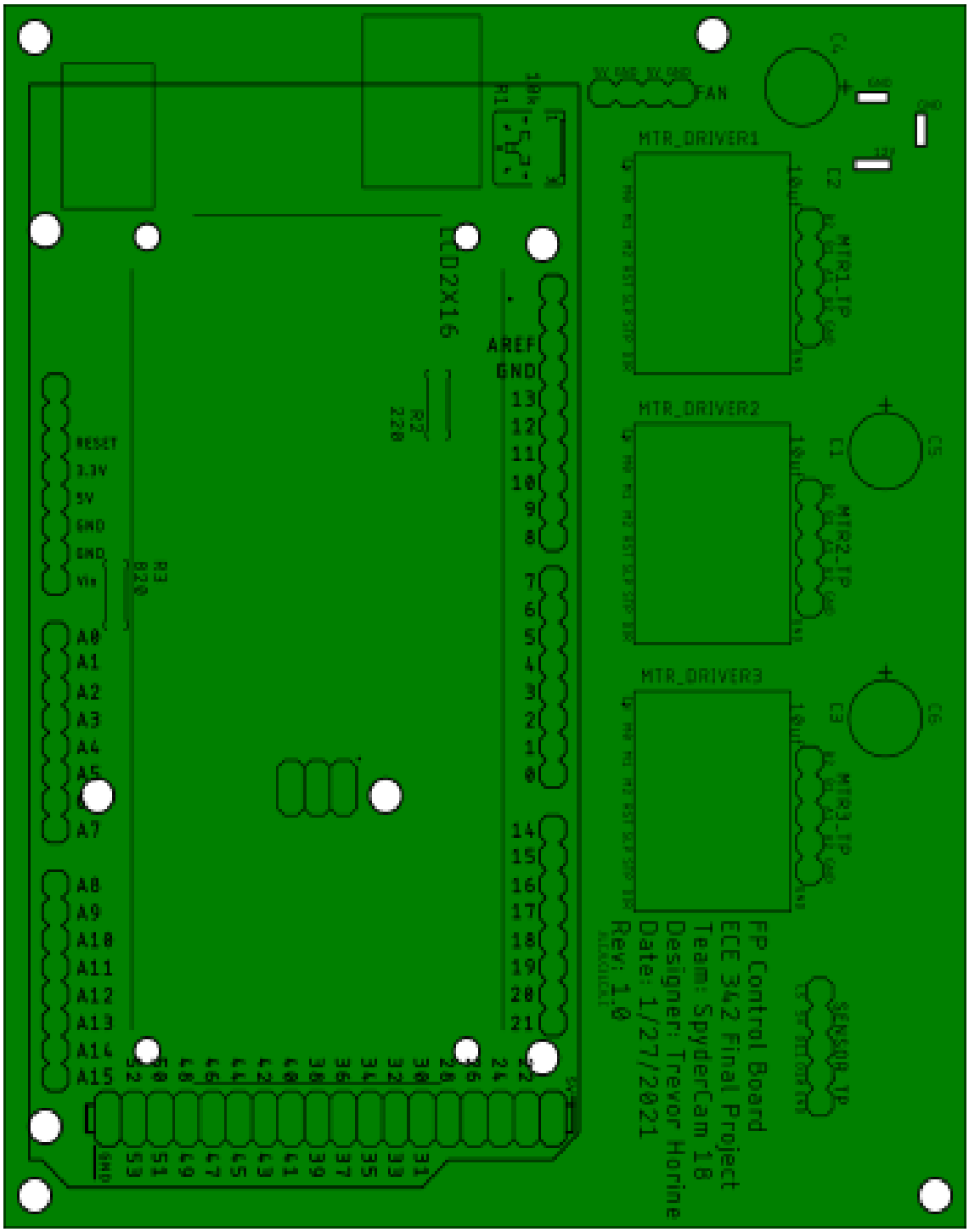

Figure 32: This is a rendering of the top silkscreen layer on top of the board profile. This image was generated from the Gerber files.

5.8 Top Copper

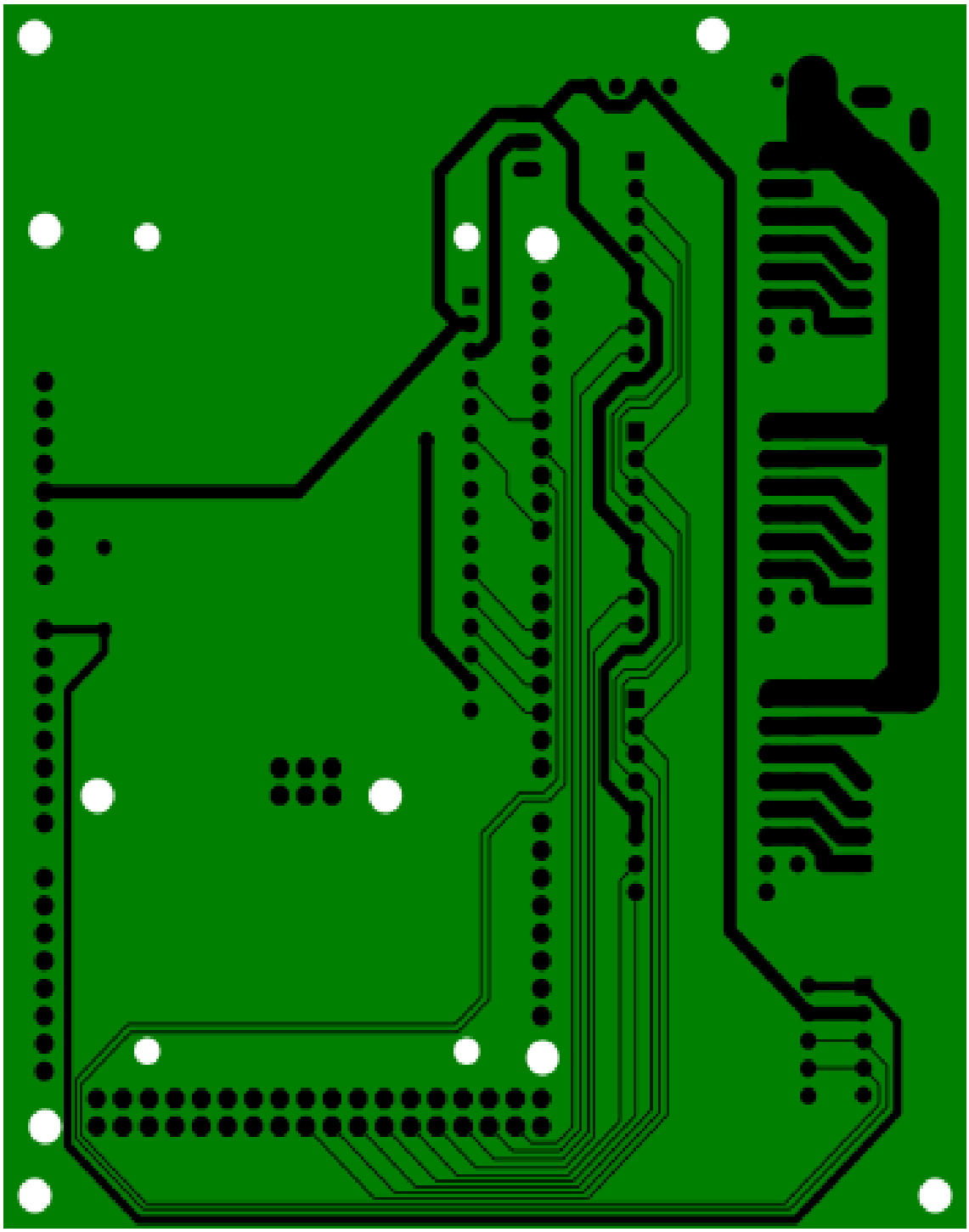

Figure 33: This is a rendering of the top copper layer on top of the board profile. This image was generated from the Gerber files.

5.9 Top Soldermask

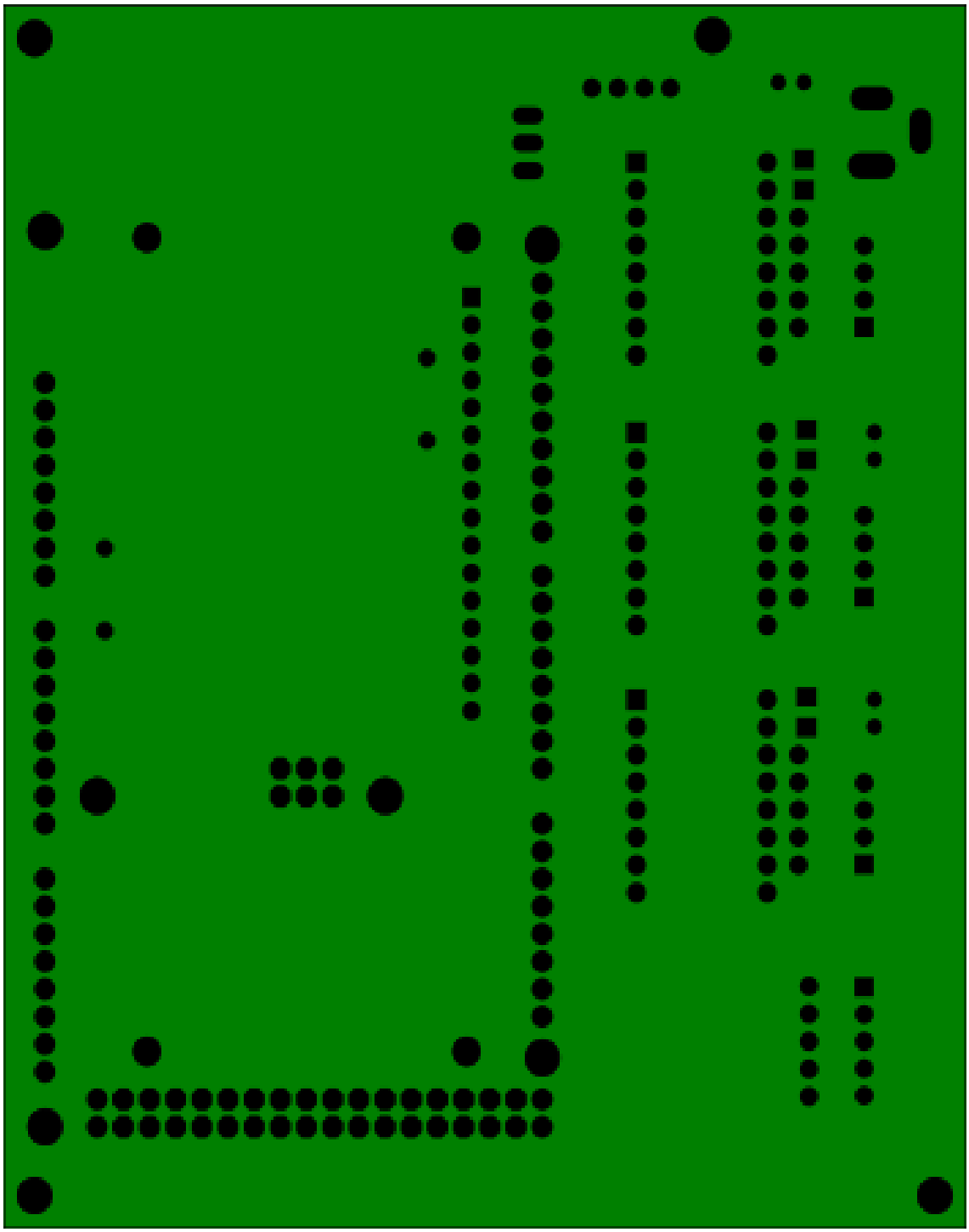

Figure 34: This is a rendering of the top soldermask layer on top of the board profile. This image was generated from the Gerber files.

5.10 Top Soldermask And Silkscreen

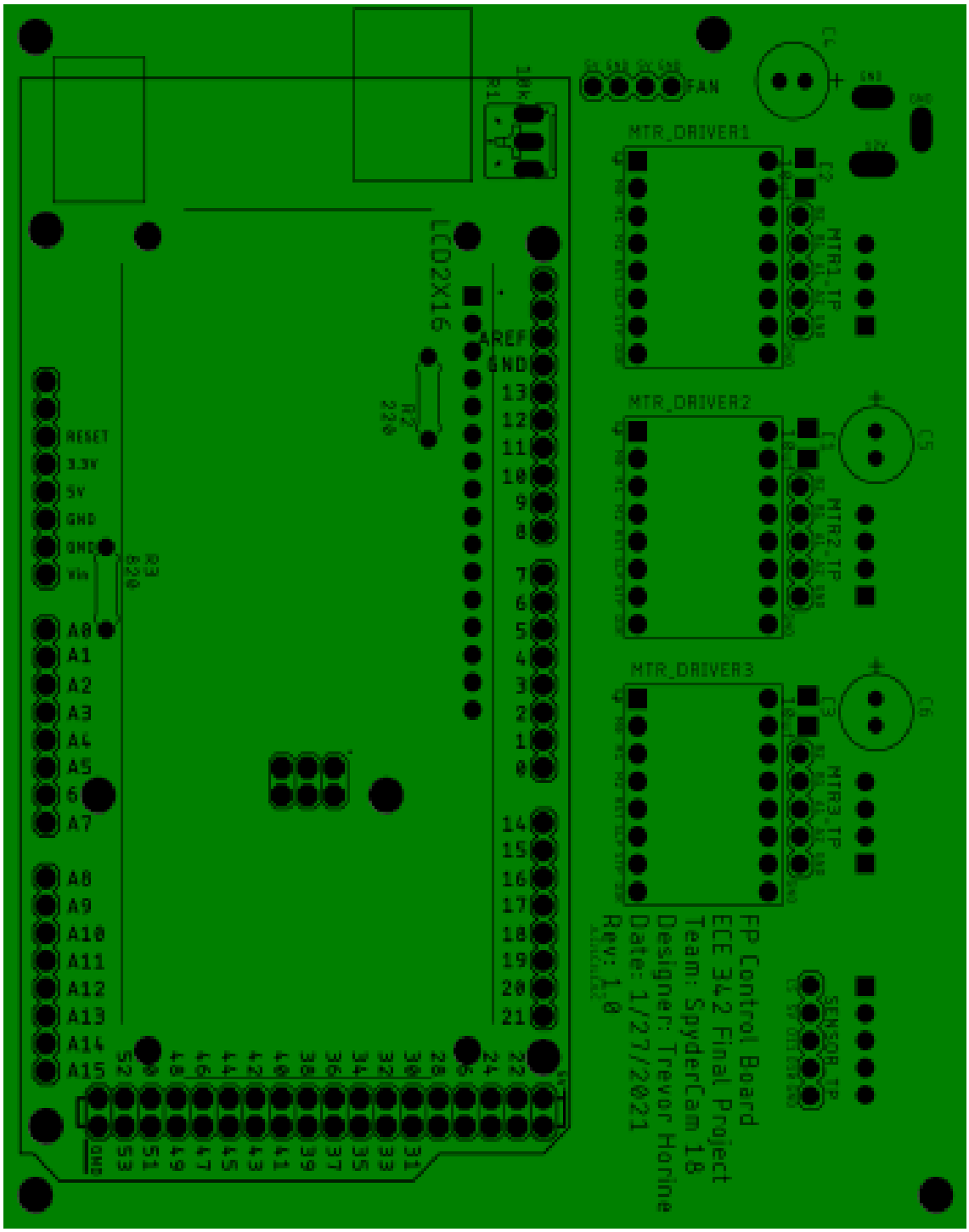

Figure 35: This is a rendering of the top soldermask layer and the top silkscreen layer on top of the board profile. This image was generated from the Gerber files.

5.11 Bottom Silkscreen

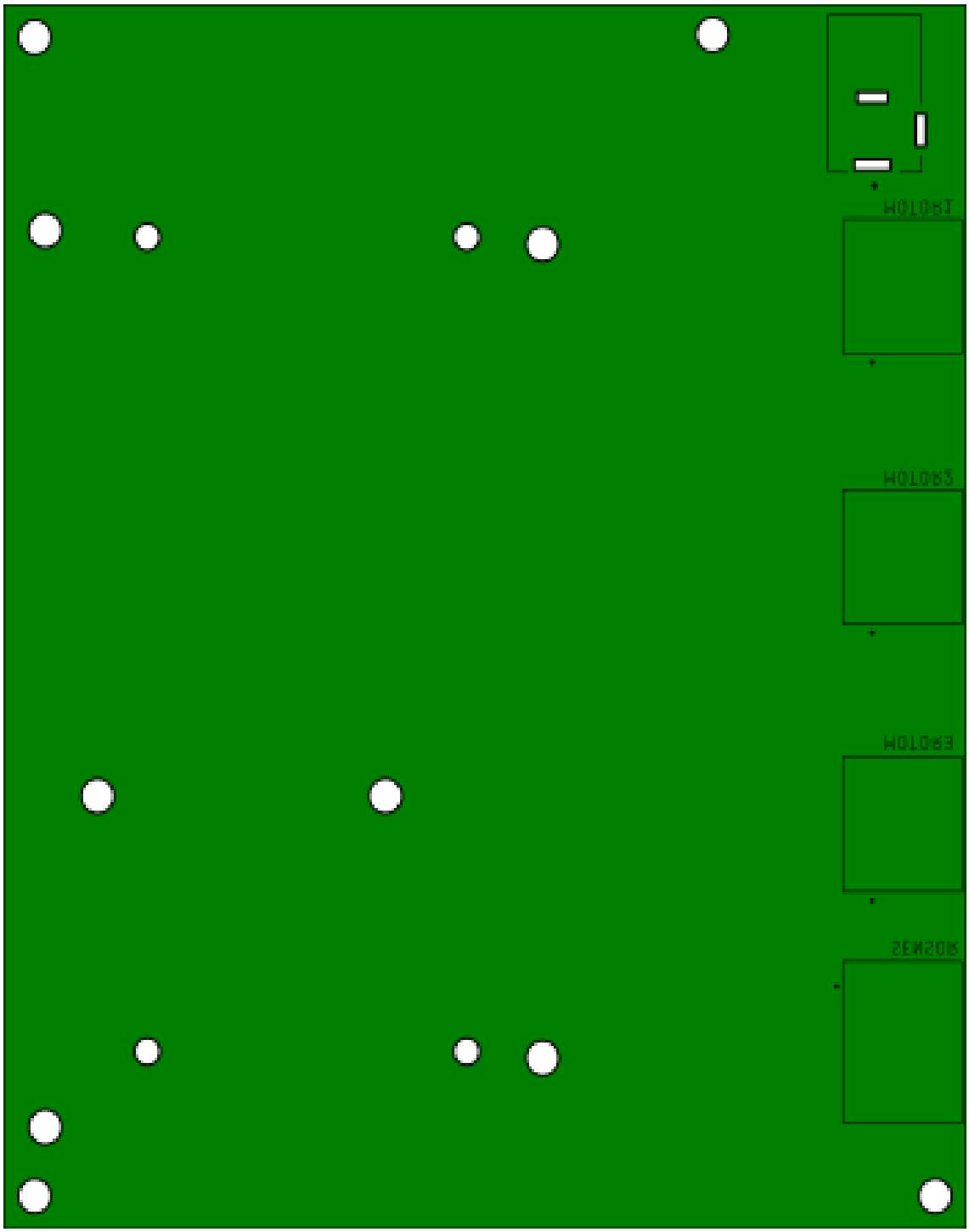

Figure 36: This is a rendering of the bottom silkscreen layer on top of the board profile. This image was generated from the Gerber files.

5.12 Bottom Copper

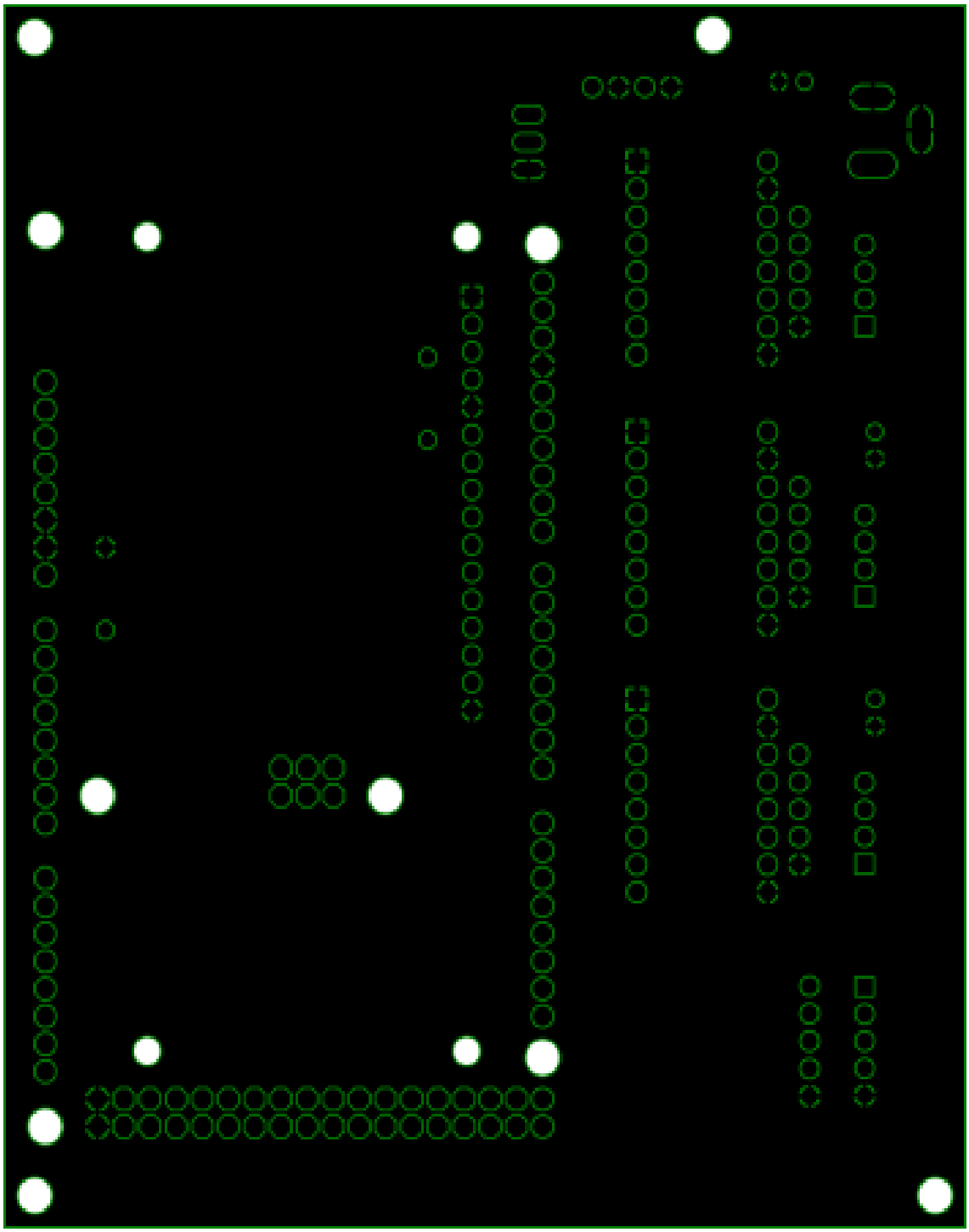

Figure 37: This is a rendering of the bottom copper layer on top of the board profile. This image was generated from the Gerber files.

5.13 Bottom Soldermask

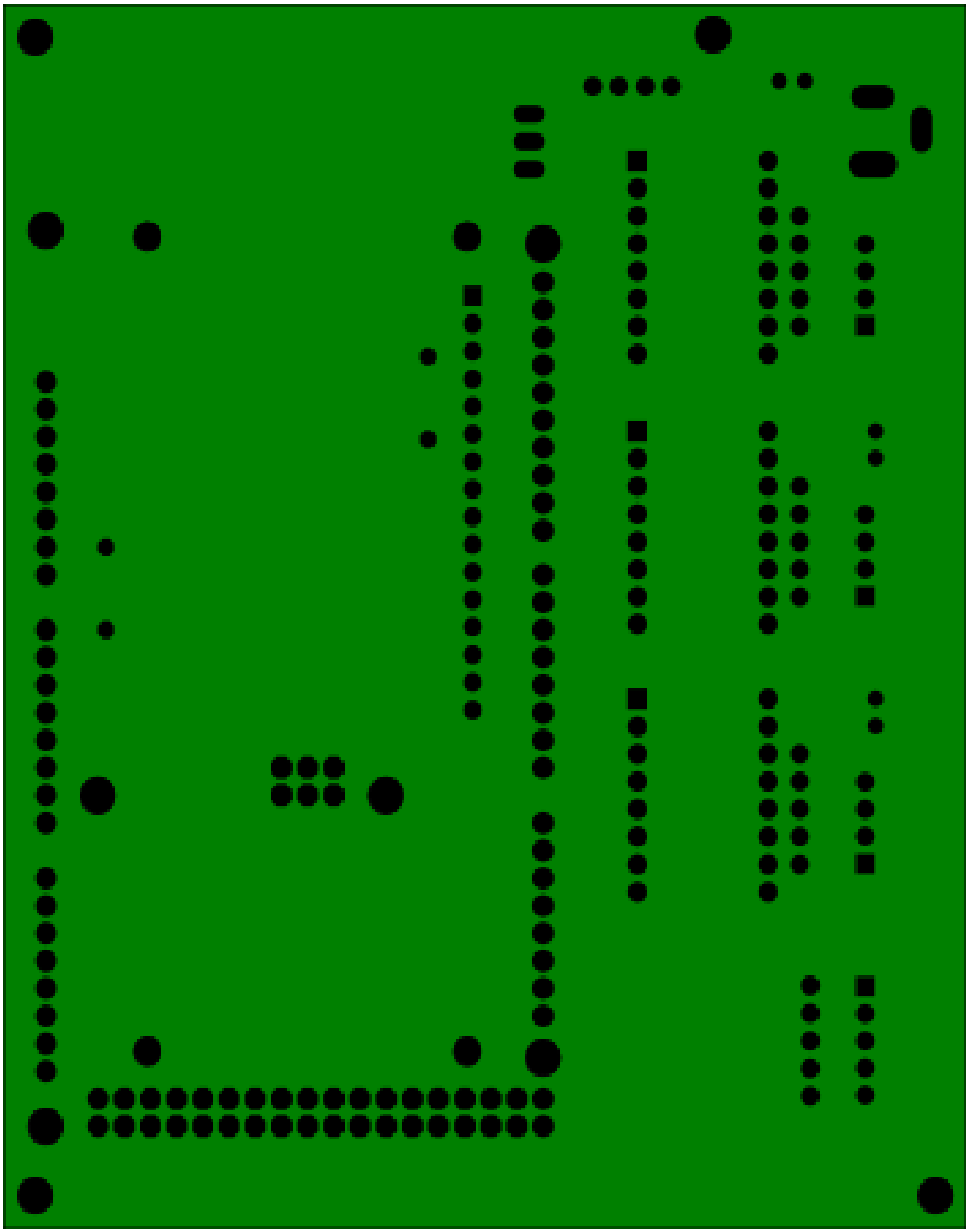

Figure 38: This is a rendering of the bottom soldermask layer on top of the board profile. This image was generated from the Gerber files.

5.14 Bottom Soldermask and Silkscreen

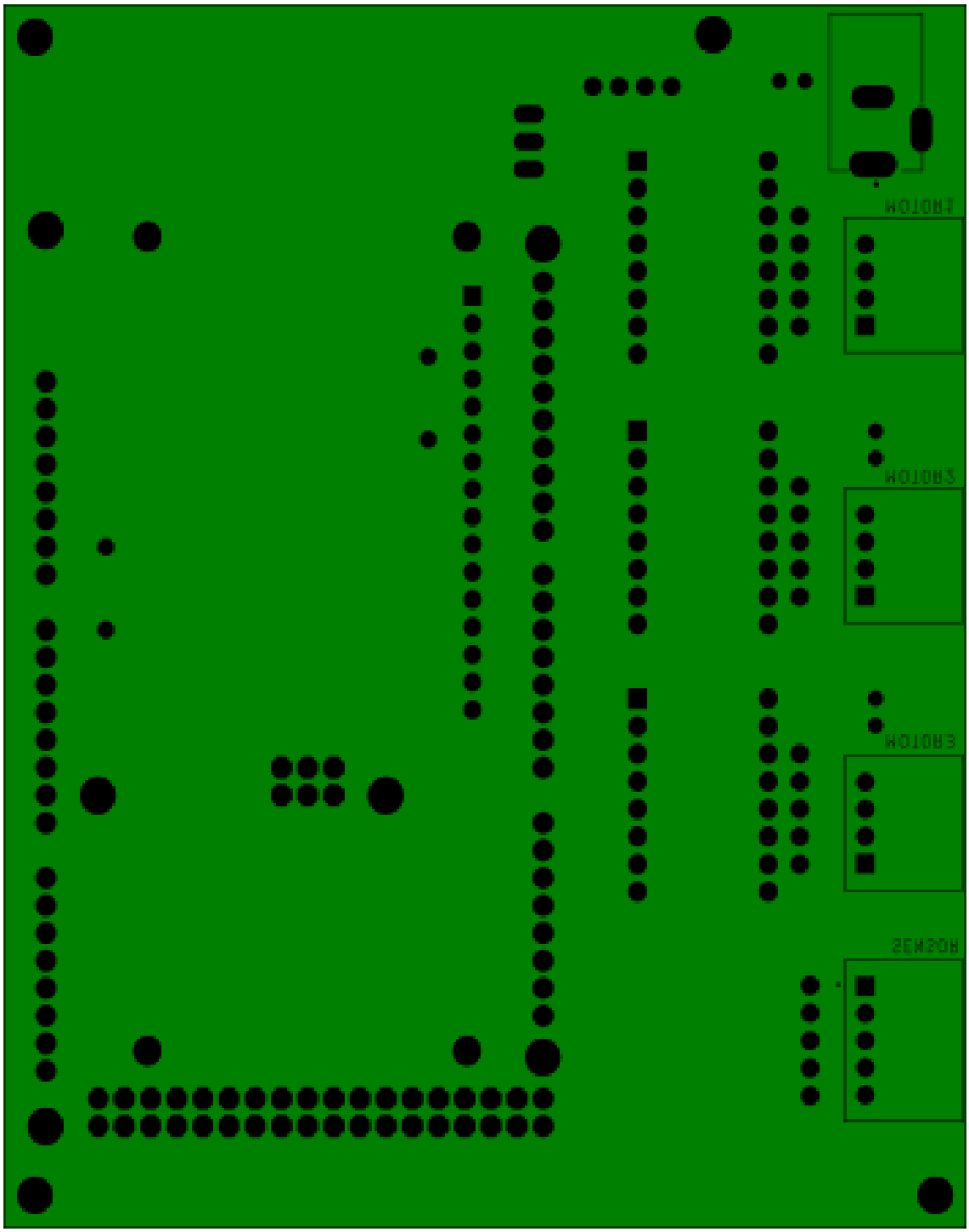

Figure 39: This is a rendering of the bottom soldermask and silkscreen layers on top of the board profile. This image was generated from the Gerber files.

5.15 Top Gerbers

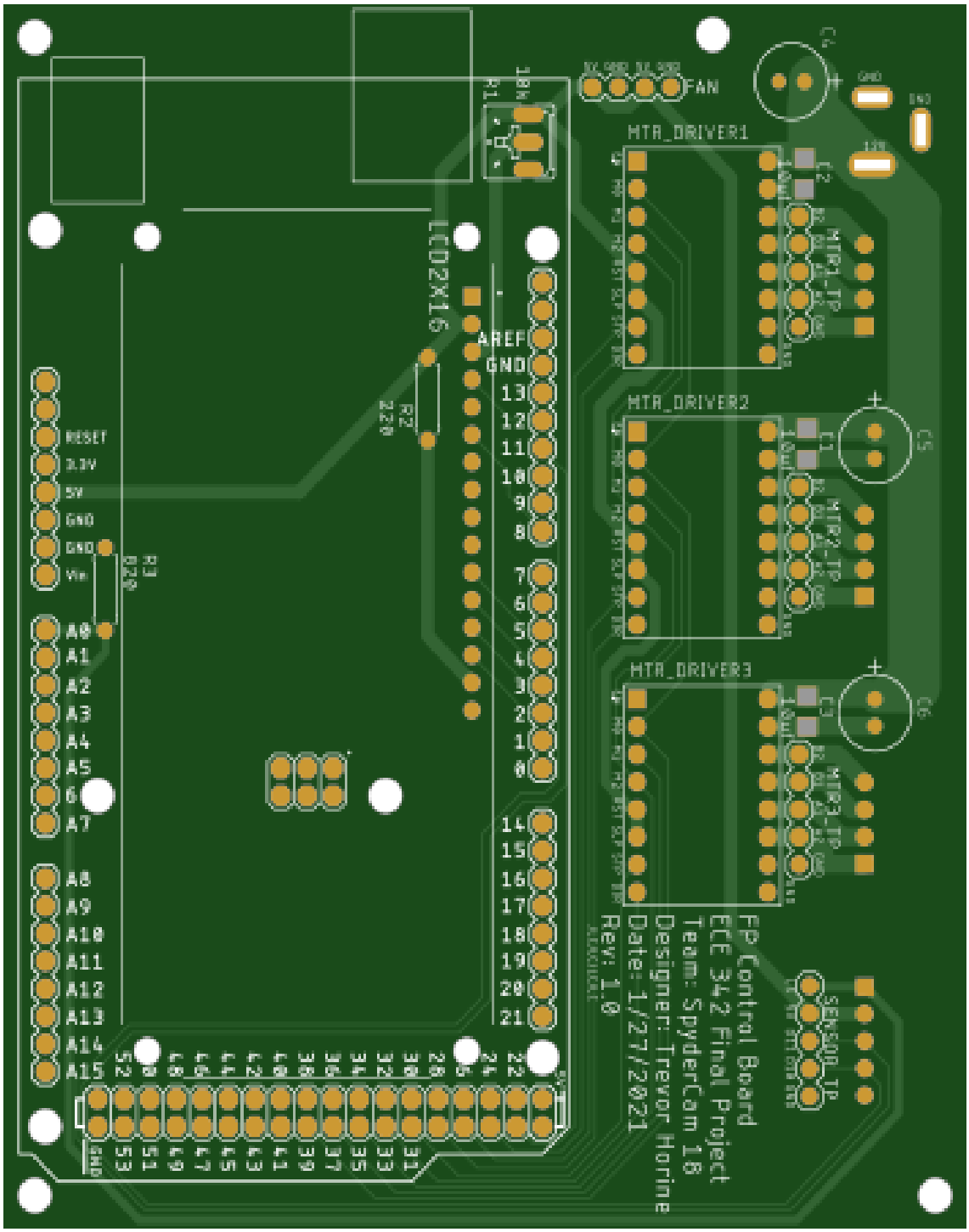

Figure 40: This is a rendering of the top layer Gerber files placed on top of each other. The white is the text and lines are the silkscreen, the gold is the soldermask, light green is the copper traces, and gray is the solderpaste. All of this is on top of the profile of the board or the darker green area, the black holes are holes in the board.

5.16 Bottom Gerbers

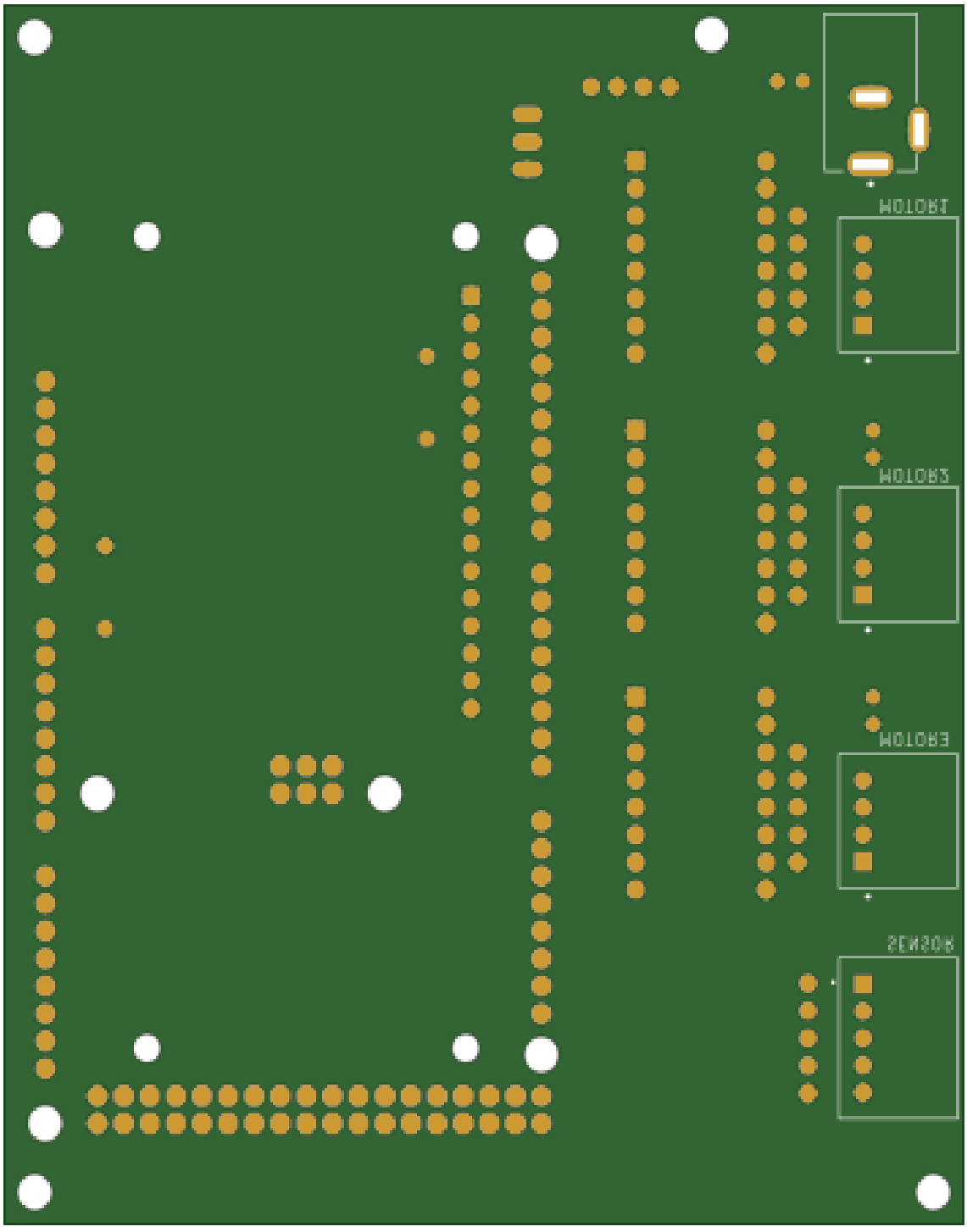

Figure 41: This is a rendering of the bottom layer Gerber files placed on top of each other. This layer has no white silkscreen, the gold is the soldermask, light green is the copper traces, and there is no gray solderpaste on this layer. All of this is placed on the profile of the board or the darker green area, the black holes are holes in the board. It may be hard to tell but the is only a little dark green around the soldermask and the edges of the board and holes because the bottom of this board is a copper ground plane.

5.17 3D Model

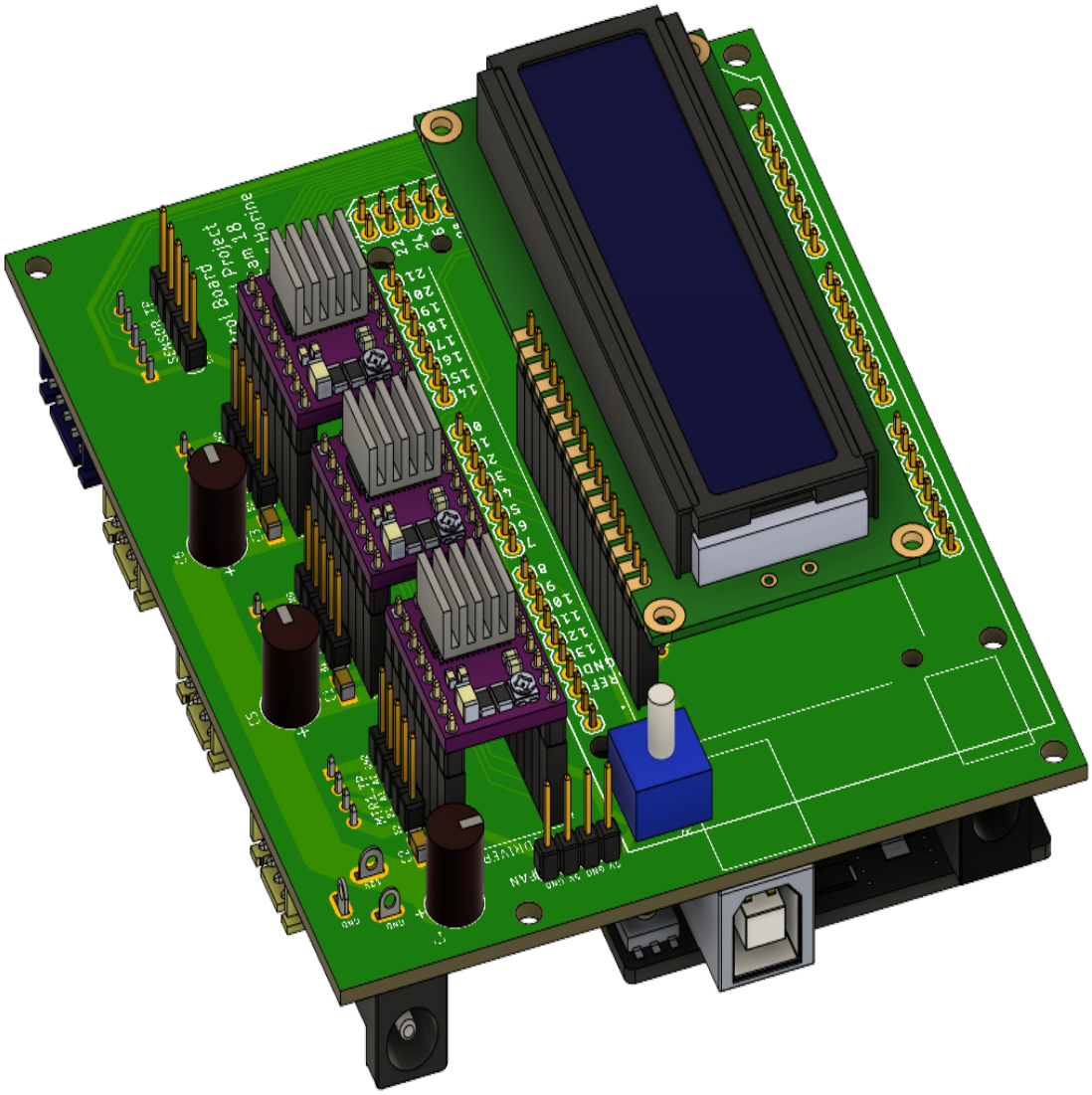

Figure 42: This is a rendering of the 3D model produced form the board designed with components added to give an idea of what the board will look like assembled. It will also be used to design the enclosure.

5.18 Assembled Board

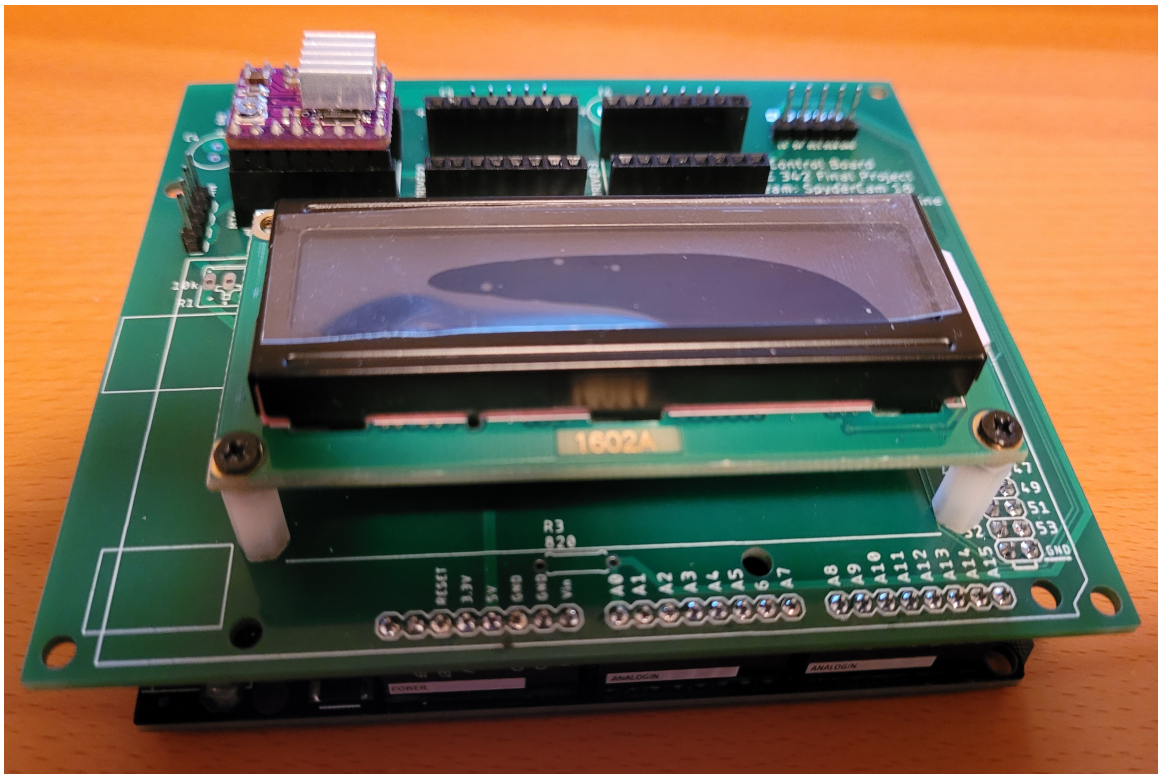

Figure 43: This is the board with some of the components added to it.

6 Parts Information and Bill of Material

Part	Value and units	Quaintity	Price	Notes
PCBs				Boards have to be ordered in incriments of 5.
Stepper drivers		5	9.99	Datasheet was found for a board that appears to be same as there was not a datasheet on the amazon page. DRV8825 Chip data sheet
Pin Headers		150	0	Will use pin headers from previous classes 64 female, 86 male
Barrel Jack		1	0.53	Used to conect wall wart to board
LCD		1	3.25	Used for debug and display
S5B-XH(LF)(SN)	Conector	1	0.31	Conector to attach light sensor and RFID 5 pin right angle conector
S4B-XH-A(LF)(SN)	Conector	3	0.81	Conectors to attach motors to board each one is a 4 pin right angle conector
Arduino Mega 2560	Arduino	1	16.99	Arduino, this is the brain
POTENTIOMETER	10k	1	2.9	Used for contrast on LCD
Resistor	220	1	0	Resistor for LCD
Resistor	10K	1	0	Resistor for voltage divider with photocell to measure light level
Capacitor	100µ	3	0.69	Used on the 12 volt power source before the motor drivers to help with current draw
Capacitor	10µ	3	0.66	Used on the 12 volt power source before the motor drivers to help with current draw. These may not be needed but pads were added so they could be added later if needed.
Fan	Fan	2	6	40mmx40mmx10mm fan should it be needed to keep motor drivers from heating up to much.
Boards		5	10	Minimum order quantitay is 5 baords.
PCB Enclosure		1	5	3D printed enclosusre for the PCB
M3, 35mm screws		4	2.48	Screws used to hold PCB in place and the eclosure colsed.
PAYLOAD				
Photocell		1	0.89	photocell to be used in voltage divider for measuering light level
RFID breakout		1	10	
Female Cable Connector Housing		1	0.12	
Ribbon Cable	3 ft, 6 wire 24awg	1	5.05	
#2 screw- RFID board mount		2	0.14	Used to mount board to payload
#6-32 1/2" set screw		1	0.19	Used to secure writing utensil
Payload Enclosure- Top		1	5.44	3D printed enclosure for the payload sensors and writing implment through Corvallis3d.
Payload Enclosure- Bottom		1	14.98	3D printed enclosure for the payload sensors and writing implment through Corvallis3d.
Mechanical				
Stepper motors		3	24.99	Datasheet info on the amazon page
Stepper Motor Brackets		5	13.99	
2" Aluminum Round Stock		1 ft	0	Used scrap aluminum
Aluminum Tube Stock 1/8" wall, 1" nominal (ID)		3.375 ft	0	Used scrap aluminum
15/32" Plywood sheet		3.5' x 3.5'	0	Used scrap wood
1/4"-20 x 1.25" Socket Head Cap Screw		9	1.71	Pylon attachment hardware
1/4" Washer		9	0.63	Pylon attachment hardware
1/4"-20 Nut		9	0.81	Pylon attachment hardware
Fishing line			0	Monofilament 9lb test fishing line was used as a strong light weight string to conect the payload to the motors.
			Total Cost	
			138.55	

Figure 44: This is the bill of materials for the system. The bill of materials also lists component's values, name on board, quantities and costs.

7 Time Report

7.1 Time sheet

Name	Date	Time spent (Hours rounded to nearist .5)	What did you work on
All	1/8/2021	4	Team Meeting
All	1/10/2021	4	Team Meeting
All	1/11/2021	4	Meet with mentor an debrief
All	1/13/2021	2	Recitation work time
Trevor	1/13/2021	1	CAD Drawings
All	1/14/2021	8	Team Meeting
Ben	1/14/2021	2	Hardware Research and order
Miles	1/12/2021	3	MATLAB Code
Trevor	1/15/2021	4	Schematics, Sensor payload, and PCB
Trevor	1/16/2021	6	Schematics, Sensor payload, and PCB
Trevor	1/17/2021	5	Schematics and PCB
Ben	1/17/2021	6	Mechanical Component Design & Fabrication
Ben	1/20/2021	2	Mechanical Component Design & Fabrication
All	1/24/2021	12	Team meeting and block checkoff prep
Miles	1/26/2021	8	MATLAB GUI
Ben	1/26/2021	3	Mechanical assembly testing
Trevor	1/26/2021	5	PCB finishing touches and ordering
Trevor	2/5/2021	3	Putting the PCBs together
Ben	2/6/2021	6	Payload Design
Trevor	2/1-9/2021	18	Enclosure design
Ben	2/8/2021	3	Payload Redesign for 3d printing
Trevor	2/11-2/13	3	Enclosure mechanical drawings
Ben	2/13/2021	4	Payload implementation and testing
Trevor	2/15/2021	1	Enclosure
Miles	2/15/2021	4	MATLAB Arduino Serial Comm
Miles	2/17/2021	4	MATLAB Arduino Serial Comm
Ben	2/18/2021	2	Hardware integration testing
Felipe	1/17/2021	12	Arduino Control Software Design/Implementation
Felipe	2/7/2021	8	Arduino Motor Control Firmware Design/Implementation
Felipe	2/14/2021	9	Testing Arduino Software
All	2/20/2021	12	Putting together the final hardware and troubleshooting
Miles	2/20/2021	3	Finishing serial interface and reading Arduino code
Ben	2/21/2021	3	Testing Hardware/software integration
Trevor	2/26/2021	2	CAD Drawings
All	2/27/2021	4	Debug and testing
Ben	2/28/2021	6	MATLAB testing, arduino testing, redesigning payload for backup
Miles	2/28/2021	2	MATLAB debugging
All	2/28/2021	8	System debugging and testing
Trevor	3/1/2021	3	Documentation
Ben	3/1/2021	1	Documentation
Trevor	3/3/2021	2	Documentation
		Total Manhours	
	Percentage per Person	202	
Trevor	33.42	67.5	
Ben	25.99	52.5	
Miles	19.06	38.5	
Felipe	21.53	43.5	

Figure 45: This is the time sheet used to track hours on this project.

7.2 Percentage By Person

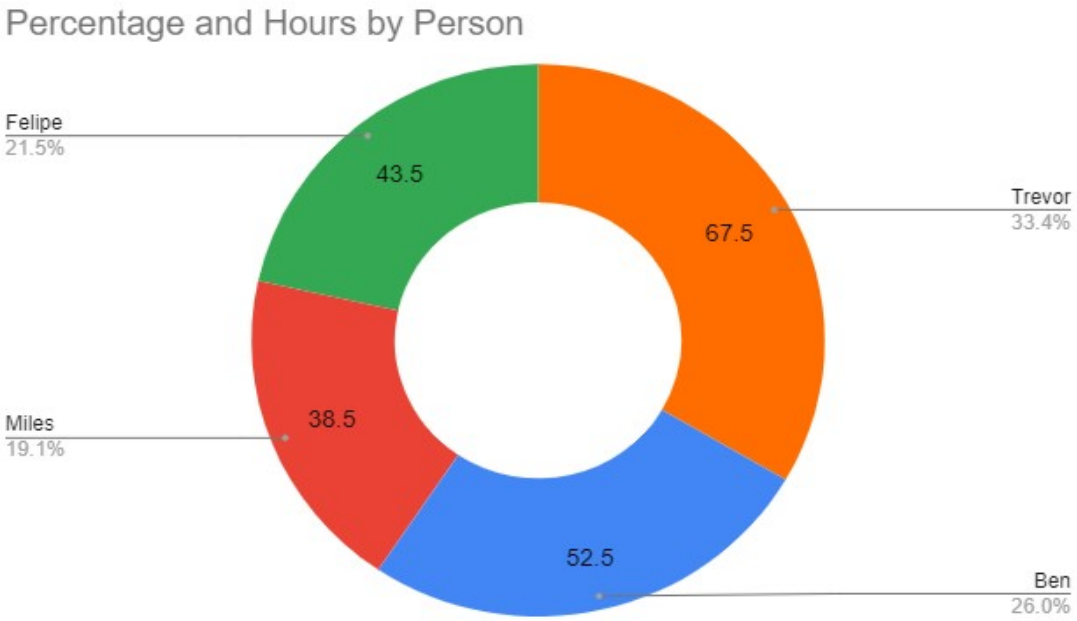

Figure 46: This graph plots the hours spent on the project by person.

7.3 What We Worked On

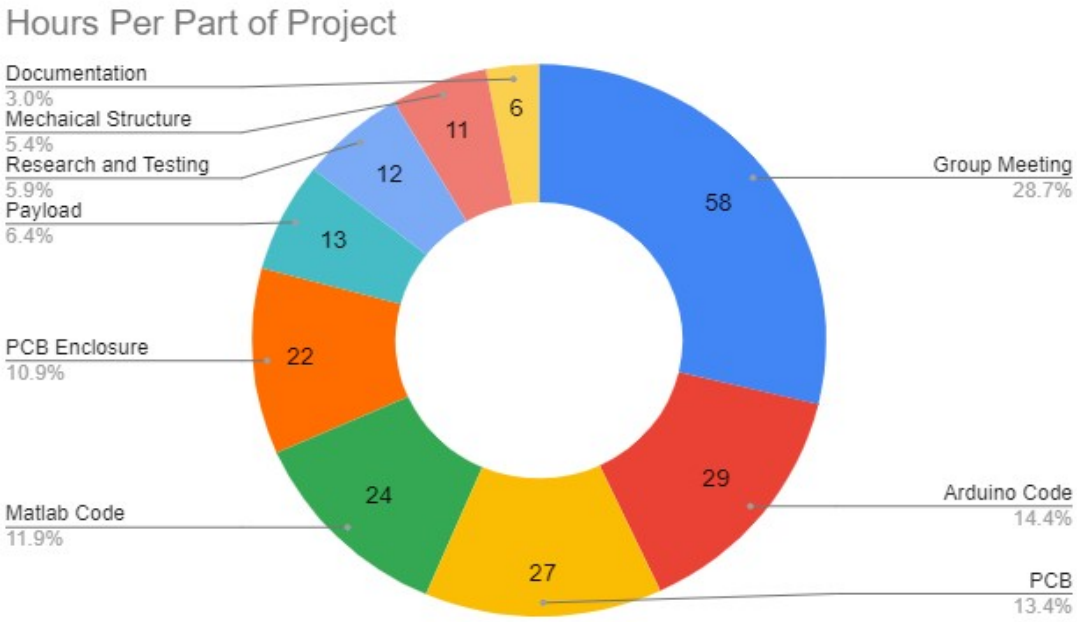

Figure 47: This graph plots the hours spent on each part of the project.

8 Appendix

8.1 Arduino Code

```
/*
  Arduino Control Software Block & Motor Control Firmware
  Author: Felipe Orrico Scognamiglio
  For: Oregon State University - Junior Design - Final Project
   Spydercam18
  Date: 03/01/2021
  Version: BETA 1.25 - FINAL
*/

/*
  Known Problems:
    - After M2 command, Incremental Coordinate mode will not work
   properly
    and C1 will not be able to report correct coordinates
    - Inputting M2 and M6 right after the other has undefined
   behaviour
    - Current X, Y, Z positions are not available after receiving
   M2/M6, only
    after another G0 or G1 is executed. This happens because it is
   hard to extrapolate
    the current position only based on current thread lengths.
   Instead, the program will
    update after another G0 or G1 command.
*/


//////////////////////////LCD SETUP//////////////////////////
#include <LiquidCrystal.h>
const int rs = 12, en = 8, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
//////////////////////////LCD SETUP//////////////////////////

//////////////////////////Sensor SETUP//////////////////////////
#include <SoftwareSerial.h>
#include "PN532_SWHSU.h"
#include "PN532.h"

SoftwareSerial SWSerial( 10, 11 ); // RX, TX
PN532_SWHSU pn532swhsu( SWSerial );
PN532 nfc( pn532swhsu );

const int sensor = A0;

float sensorVal = 0;
float voltage = 0;
float lightLevel = 0;

//////////////////////////Sensor SETUP//////////////////////////


//////////////////////////CONSTANTS//////////////////////////

//ALL PHYSICAL VALUES ARE REPORTED IN INCHES
const double A_PAPER_DISTANCE = 11.583;
```

```
const double B_PAPER_DISTANCE = 7.2;
const double C_PAPER_DISTANCE = 7.2;

const double HEIGHT_PYLON_A = 11.8;
const double HEIGHT_PYLON_B = 11.8;
const double HEIGHT_PYLON_C = 11.8;

const double PAYLOAD_CENTER_THREAD_DISTANCE_A_x = 3.6;
const double PAYLOAD_CENTER_THREAD_DISTANCE_C_x = 1.85;
const double PAYLOAD_CENTER_THREAD_DISTANCE_C_y = 3.15;
const double PAYLOAD_CENTER_THREAD_DISTANCE_B_x = 1.85;
const double PAYLOAD_CENTER_THREAD_DISTANCE_B_y = 3.15;

const double PAPER_HEIGHT = 11.00;
const double PAPER_WIDTH = 8.50;

const double PAPER_DISTANCE_LOW_EDGE = 1.856;

const double HOME_X = 0;
const double HOME_Y = 0;
const double HOME_Z = 6;

const double INCHES_PER_STEP = 0.024662;
const double MAX_STEPS_PER_SEC = 182;

//////////////////////////CONSTANTS//////////////////////////

//////////////////////////VARIABLES//////////////////////////

double CURRENT_PAYLOAD_HEIGHT = 0;
double CURRENT_THREAD_LEN_A = 0;
double CURRENT_THREAD_LEN_B = 0;
double CURRENT_THREAD_LEN_C = 0;
double CURRENT_X = 0;
double CURRENT_Y = 0;

//values that are updated when there is movement
int OPCODE = -1;
double X_ADDRESS_TARGET = -1;
double Y_ADDRESS_TARGET = -1;
double Z_ADDRESS_TARGET = -1;
int F_PERCENT_SPEED = 100;

///M6///
double M6_SAVE_LEN_A = 0;
double M6_SAVE_LEN_B = 0;
double M6_SAVE_LEN_C = 0;
double M6_SAVE_X = -1;
double M6_SAVE_Y = -1;
double M6_SAVE_Z = -1;
double M6_SAVE_Speed = 0;
bool M6_EN = false;

//////////////////////////VARIABLES//////////////////////////

////////////////////////////FLAGS////////////////////////////

bool UNIT_MODE = true; //true = inches, false = mm;
```

```
107 bool ABSOLUTE_COORDINATES = true; //true = absolute, false =
    incremental
108 bool ON_MOVE = false;
109 bool M2_EN = false;
110
111 ////////////////////////////FLAGS////////////////////////////
112
113 ////////////////////////////STEPPER////////////////////////////
114
115 #include "AccelStepper.h"
116 #include "MultiStepper.h"
117
118 //change pins here to reflect correct pins
119 #define dirPinA 25
120 #define stepPinA 23
121 #define dirPinB 29
122 #define stepPinB 27
123 #define dirPinC 33
124 #define stepPinC 31
125
126 #define motorInterfaceType 1
127
128 AccelStepper stepperA;// = AccelStepper(motorInterfaceType,
    stepPinA, dirPinA);
129 AccelStepper stepperB;// = AccelStepper(motorInterfaceType,
    stepPinB, dirPinB);
130 AccelStepper stepperC;// = AccelStepper(motorInterfaceType,
    stepPinC, dirPinC);
131
132 MultiStepper steppers = MultiStepper();
133
134 ////////////////////////////STEPPER////////////////////////////
135
136 ////////////////////////G-CODE LIB SETUP//////////////////////
137 #include "gcode.h"
138 #include "Arduino_Firmware_ALPHA.h"
139 #define NumberOfCommands 10
140 //you can add more more commands here if needed and create a
    function under G-Code to execute.
141 struct commandscallback commands[NumberOfCommands] = {{"G0",G0_cmd
    },{"G1",G1_cmd},{"G20",G20_cmd},{"G21",G21_cmd},{"G90",G90_cmd
    },{"G91",G91_cmd},{"M2",M2_cmd},{"M6",M6_cmd},{"C1",C1_cmd},{"C2
    ",C2_cmd}};
142 gcode Commands(NumberOfCommands,commands);
143 ////////////////////////G-CODE LIB SETUP//////////////////////
144
145 ////////////////////////////G-Code////////////////////////////
146
147 void G0_cmd(){
148
149       if (ON_MOVE || M6_EN) {
150         //Serial.println("ON_MOVE");
151         return;
152       }
153
154
155       double x_coord = CURRENT_X;
156       double y_coord = CURRENT_Y;
```

```
157       double z_coord = CURRENT_PAYLOAD_HEIGHT;
158
159       if(Commands.availableValue('X')){
160         x_coord = Commands.GetValue('X');
161       }
162       if(Commands.availableValue('Y')){
163         y_coord = Commands.GetValue('Y');
164       }
165       if(Commands.availableValue('Z')){
166         z_coord = Commands.GetValue('Z');
167       }
168
169       int f_speed = 100; //maximum speed
170
171       if (!UNIT_MODE){ //translate mm to in
172         x_coord = mm_to_in(x_coord);
173         y_coord = mm_to_in(y_coord);
174         z_coord = mm_to_in(z_coord);
175       }
176
177       if (!ABSOLUTE_COORDINATES){//incremental mode
178         x_coord += CURRENT_X;
179         y_coord += CURRENT_Y;
180         z_coord += CURRENT_PAYLOAD_HEIGHT;
181       }
182
183       //checking for boundaries
184       if (x_coord > 8.5)
185         x_coord = 8.5;
186       if (y_coord > 11)
187         x_coord = 11;
188       if (z_coord > 11)
189         z_coord = 11;
190
191       //update global values
192       Y_ADDRESS_TARGET = y_coord;
193       X_ADDRESS_TARGET = x_coord;
194       Z_ADDRESS_TARGET = z_coord;
195       F_PERCENT_SPEED = f_speed;
196
197       update_lcd(0);
198
199       go(x_coord,y_coord,z_coord, f_speed);
200 }
201
202 void G1_cmd(){
203
204       if (ON_MOVE || M6_EN) return;
205
206       double x_coord = CURRENT_X;
207       double y_coord = CURRENT_Y;
208       double z_coord = CURRENT_PAYLOAD_HEIGHT;
209       double f_speed_in = 4.5;
210       int f_speed = 100; //maximum speed
211
212       if(Commands.availableValue('X')){
213         x_coord = Commands.GetValue('X');
214       }
```

```
215       if(Commands.availableValue('Y')){
216         y_coord = Commands.GetValue('Y');
217       }
218       if(Commands.availableValue('Z')){
219         z_coord = Commands.GetValue('Z');
220       }
221       if(Commands.availableValue('F')){
222         f_speed_in = Commands.GetValue('F');
223       }
224
225       if (!UNIT_MODE){ //translate mm to in
226         x_coord = mm_to_in(x_coord);
227         y_coord = mm_to_in(y_coord);
228         z_coord = mm_to_in(z_coord);
229         f_speed_in = mm_to_in(f_speed_in);
230       }
231
232       if (!ABSOLUTE_COORDINATES){//incremental mode
233         x_coord += CURRENT_X;
234         y_coord += CURRENT_Y;
235         z_coord += CURRENT_PAYLOAD_HEIGHT;
236       }
237
238       //checking for boundaries
239       if (x_coord > 8.5)
240         x_coord = 8.5;
241       if (y_coord > 11)
242         x_coord = 11;
243       if (z_coord > 11)
244         z_coord = 11;
245
246      f_speed_in = (f_speed_in/4.5)*100;
247
248       if (f_speed_in > 100)
249         f_speed = 100;
250       else if (f_speed_in <= 1)
251         f_speed = 1;
252       else
253         f_speed = int(f_speed_in);
254
255       //update global values
256       Y_ADDRESS_TARGET = y_coord;
257       X_ADDRESS_TARGET = x_coord;
258       Z_ADDRESS_TARGET = z_coord;
259       F_PERCENT_SPEED = f_speed;
260
261       update_lcd(1);
262
263       go(x_coord, y_coord, z_coord, f_speed);
264 }
265
266 void G20_cmd(){
267
268   if (ON_MOVE || M6_EN) return;
269
270   UNIT_MODE = true;
271   update_lcd(20);
272 }
```

```
273
274 void G21_cmd(){
275
276   if (ON_MOVE || M6_EN) return;
277
278   UNIT_MODE = false;
279   update_lcd(21);
280 }
281
282 void G90_cmd(){
283
284   if (ON_MOVE || M6_EN) return;
285
286   ABSOLUTE_COORDINATES = true;
287   update_lcd(90);
288 }
289
290 void G91_cmd(){
291
292   if (ON_MOVE || M6_EN) return;
293
294   ABSOLUTE_COORDINATES = false;
295   update_lcd(91);
296 }
297
298 void M2_cmd(){
299   update_lcd(2);
300   if (ON_MOVE){
301       stepperA.stop();
302       stepperB.stop();
303       stepperC.stop();
304
305       ON_MOVE = false;
306       ABSOLUTE_COORDINATES = true;
307       M2_EN = true;
308       M6_EN = false;
309   }
310 }
311
312 void M6_cmd(){
313   update_lcd(6);
314   if (!M6_EN){ //m6 first time
315     //stop
316
317     //if (ON_MOVE){
318     stepperA.stop();
319     stepperB.stop();
320     stepperC.stop();
321
322     ON_MOVE = false;
323     ABSOLUTE_COORDINATES = true;
324     M6_EN = true;
325
326     double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
327     double actual_b = -(stepperB.currentPosition() *
    INCHES_PER_STEP);
328     double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;
329
```

```
330     CURRENT_THREAD_LEN_A += actual_a;
331     CURRENT_THREAD_LEN_B += actual_b;
332     CURRENT_THREAD_LEN_C += actual_c;
333 
334     M6_SAVE_LEN_A = CURRENT_THREAD_LEN_A;
335     M6_SAVE_LEN_B = CURRENT_THREAD_LEN_B;
336     M6_SAVE_LEN_C = CURRENT_THREAD_LEN_C;
337     M6_SAVE_X = X_ADDRESS_TARGET;
338     M6_SAVE_Y = Y_ADDRESS_TARGET;
339     M6_SAVE_Z = Z_ADDRESS_TARGET;
340 
341     if (M6_SAVE_X == -1 || M6_SAVE_Y == -1 || M6_SAVE_Z == -1){
342       M6_SAVE_X = CURRENT_X;
343       M6_SAVE_Y = CURRENT_Y;
344       M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
345     }
346     if (M6_SAVE_X == 0 && M6_SAVE_Y == 0 && M6_SAVE_Z == 0){
347       M6_SAVE_X = CURRENT_X;
348       M6_SAVE_Y = CURRENT_Y;
349       M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
350     }
351     M6_SAVE_Speed = F_PERCENT_SPEED;
352 
353     go_block(HOME_X,HOME_Y,HOME_Z, 100);
354   }
355   else {
356 
357     go_len_block(M6_SAVE_LEN_A,M6_SAVE_LEN_B,M6_SAVE_LEN_C, 100);
358 
359     M6_EN = false;
360 
361     //go(M6_SAVE_X,M6_SAVE_Y,M6_SAVE_Z,M6_SAVE_Speed);
362 
363     /*M6_SAVE_LEN_A = 0;
364     M6_SAVE_LEN_B = 0;
365     M6_SAVE_LEN_C = 0;
366     M6_SAVE_X = -1;
367     M6_SAVE_Y = -1;
368     M6_SAVE_Z = -1;
369     M6_SAVE_Speed = 0;*/
370   }
371 }
372 
373 void C1_cmd(){ //Write value of X,Y,Z to the Serial Connection
374 
375   //if (ON_MOVE || M6_EN) return;
376 
377   Serial.write("X");
378   Serial.print(CURRENT_X);
379   Serial.write("Y");
380   Serial.print(CURRENT_Y);
381   Serial.write("Z");
382   Serial.print(CURRENT_PAYLOAD_HEIGHT);
383   Serial.write("#");
384   update_lcd(57);
385   delay(500);
386 }
387 
```

```
388 void C2_cmd() { //Write thread lengths to Serial
389   //if (ON_MOVE || M6_EN) return;
390 
391   Serial.write("A");
392   Serial.print(CURRENT_THREAD_LEN_A);
393   Serial.write("B");
394   Serial.print(CURRENT_THREAD_LEN_B);
395   Serial.write("C");
396   Serial.print(CURRENT_THREAD_LEN_C);
397   Serial.write("#");
398   update_lcd(58);
399   delay(500);
400 }
401 
402 ////////////////////////////G-Code//////////////////////////////
403 
404 
405 ////////////////////THREAD CALCULATIONS////////////////////
406 double calculate_thread_height_var(double z){
407     return HEIGHT_PYLON_A - z;
408 }
409 
410 double calculate_thread_length_A_h(double x, double y, double z) {
411   double height_created_triangle =  A_PAPER_DISTANCE + 8.5 - x -
      PAYLOAD_CENTER_THREAD_DISTANCE_A_x;
412   double side_created_triangle;
413   if ((y - 5.5) < 0)
414     side_created_triangle = 5.5 - y;
415   else
416     side_created_triangle = y - 5.5;
417 
418   double trace_length = sqrt(pow(height_created_triangle, 2) + pow(
      side_created_triangle, 2));
419 
420   double thread_length = sqrt(pow(trace_length,2) + pow(
      calculate_thread_height_var(z),2));
421 
422   return thread_length;
423 }
424 
425 double calculate_thread_length_B_h(double x, double y, double z) {
426   double height_created_triangle =  PAPER_HEIGHT + B_PAPER_DISTANCE
       - y - PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
427   double side_created_triangle =  PAPER_DISTANCE_LOW_EDGE + x -
      PAYLOAD_CENTER_THREAD_DISTANCE_B_x;
428 
429   double trace_length = sqrt(pow(height_created_triangle,2) + pow(
      side_created_triangle,2));
430 
431   //double trace_length = PAPER_DISTANCE_1 - y -
      PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
432 
433   double thread_length = sqrt(pow(trace_length,2) + pow(
      calculate_thread_height_var(z),2));
434 
435   return thread_length;
436 }
437 
```

```
438 double calculate_thread_length_C_h(double x, double y, double z) {
439   double height_created_triangle = C_PAPER_DISTANCE + y -
       PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
440   double side_created_triangle = PAPER_DISTANCE_LOW_EDGE + x -
       PAYLOAD_CENTER_THREAD_DISTANCE_C_x;
441
442   double trace_length = sqrt(pow(height_created_triangle,2) + pow(
       side_created_triangle,2));
443
444   double thread_length = sqrt(pow(trace_length,2) + pow(
       calculate_thread_height_var(z),2));
445
446   return thread_length;
447 }
448
449 ////////////////////THREAD CALCULATIONS////////////////////
450
451 ////////////////////UNIT CONVERSION////////////////////////
452 double mm_to_in(double mm){
453   return (mm/(25.4));
454 }
455
456 double in_to_mm(double in){
457   return (in *(25.4));
458 }
459 ////////////////////UNIT CONVERSION////////////////////////
460
461 void update_lcd(int opcode){
462   lcd.clear();
463   if (opcode == 0 || opcode == 1){ //G0 or G1 (displays as integer
       for size)
464     lcd.setCursor(0, 0); //1st line 1st char
465     lcd.print("OPCODE: G");
466     lcd.print(opcode);
467     lcd.print(" F:");
468     lcd.print(F_PERCENT_SPEED);
469     lcd.setCursor(0, 1); //2nd Line 1st char
470     double x = X_ADDRESS_TARGET;
471     double y = Y_ADDRESS_TARGET;
472     if (!UNIT_MODE){ //translate mm to in
473         x = in_to_mm(X_ADDRESS_TARGET);
474         y = in_to_mm(Y_ADDRESS_TARGET);
475     }
476     lcd.print("X: ");
477     lcd.print(x);
478     lcd.setCursor(8, 1); //2nd Line 9th char
479     lcd.print("Y: ");
480     lcd.print(y);
481   }
482   else if (opcode == 20){
483     lcd.setCursor(0, 0); //1st line 1st char
484     lcd.print("OPCODE: G");
485     lcd.print(opcode);
486     lcd.setCursor(0, 1); //2nd Line 1st char
487     lcd.print("Input Mode > IN");
488   }
489   else if (opcode == 21){
490     lcd.setCursor(0, 0); //1st line 1st char
```

```
491     lcd.print("OPCODE: G");
492     lcd.print(opcode);
493     lcd.setCursor(0, 1); //2nd Line 1st char
494     lcd.print("Input Mode > MM");
495   }
496   else if (opcode == 90){
497     lcd.setCursor(0, 0); //1st line 1st char
498     lcd.print("OPCODE: G");
499     lcd.print(opcode);
500     lcd.setCursor(0, 1); //2nd Line 1st char
501     lcd.print("Coord Mode > ABS");
502   }
503   else if (opcode == 91){
504     lcd.setCursor(0, 0); //1st line 1st char
505     lcd.print("OPCODE: G");
506     lcd.print(opcode);
507     lcd.setCursor(0, 1); //2nd Line 1st char
508     lcd.print("Coord Mode > INC");
509   }
510   else if (opcode == 2){
511     lcd.setCursor(0, 0); //1st line 1st char
512     lcd.print("OPCODE: M");
513     lcd.print(opcode);
514     lcd.setCursor(0, 1); //2nd Line 1st char
515     lcd.print("END PROGRAM");
516   }
517   else if (opcode == 6){
518     lcd.setCursor(0, 0); //1st line 1st char
519     lcd.print("OPCODE: G");
520     lcd.print(opcode);
521     lcd.setCursor(0, 1); //2nd Line 1st char
522     lcd.print("TOOL CHANGE");
523   }
524   else if (opcode == 57){
525     lcd.setCursor(0, 0); //1st line 1st char
526     lcd.print("Sending Location");
527     lcd.setCursor(0, 1); //2nd Line 1st char
528     lcd.print("<<<<<<<<>>>>>>>>");
529   }
530   else if (opcode == 58){
531     lcd.setCursor(0, 0); //1st line 1st char
532     lcd.print("Sending Threads");
533     lcd.setCursor(0, 1); //2nd Line 1st char
534     lcd.print("<<<<<<<<>>>>>>>>");
535   }
536   else {
537     lcd.setCursor(0, 0); //1st line 1st char
538     lcd.print("ERROR: ");
539     lcd.print(opcode);
540     lcd.setCursor(0, 1); //2nd Line 1st char
541     lcd.print("OPCODE NOT FOUND");
542   }
543 }
544
545 //Calculates the number of steps to move from current_ to new_
      length of thread.
546 //If the output is negative, then the motor must reduce the size of
      the thread instead of increasing it
```

```
547
548 //////////////////////////CALCULATIONS FOR MOVEMENT
      //////////////////////////
549 int calculate_number_steps(double current_, double new_){
550   int num_steps = int((new_ - current_)/INCHES_PER_STEP);
551   return num_steps;
552 }
553
554 int go_block(double x, double y, double z, int f_speed) {
555   X_ADDRESS_TARGET = x;
556   Y_ADDRESS_TARGET = y;
557   Z_ADDRESS_TARGET = z;
558   F_PERCENT_SPEED = f_speed;
559
560   //calculate new necessary thread lengths
561   double new_len_a = calculate_thread_length_A_h(x,y,z);
562   double new_len_b = calculate_thread_length_B_h(x,y,z);
563   double new_len_c = calculate_thread_length_C_h(x,y,z);
564
565   //calculate number of steps and direction of movement to move
       from current pos to new pos.
566   int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A,
       new_len_a);
567   int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B,
       new_len_b);
568   int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C,
       new_len_c);
569
570
571   //Setting important info in stepper classes
572   stepperA.setCurrentPosition(0);
573   stepperB.setCurrentPosition(0);
574   stepperC.setCurrentPosition(0);
575   int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
576   stepperA.setMaxSpeed(max_speed);
577   stepperB.setMaxSpeed(max_speed);
578   stepperC.setMaxSpeed(max_speed);
579   //move the payload to location
580   long steps[] = {steps_A, -steps_B, steps_C};
581
582   lcd.clear();
583   lcd.print("Moving to >>>>");
584   lcd.setCursor(0, 1);
585   lcd.print(x);
586   lcd.print(" ");
587   lcd.print(y);
588   lcd.print(" ");
589   lcd.print(z);
590
591   steppers.moveTo(steps);
592
593   //move_to(steps_A, steps_B, steps_C);
594
595   //blocks until steppers finish moving
596   while(steppers.run()){
597   }
598
599   //find actual displacement
```

```
600   double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
601   double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP)
       ;
602   double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;
603
604   //after done, update current values
605   CURRENT_X = x;
606   CURRENT_Y = y;
607   CURRENT_PAYLOAD_HEIGHT = z;
608   CURRENT_THREAD_LEN_A += actual_a;
609   CURRENT_THREAD_LEN_B += actual_b;
610   CURRENT_THREAD_LEN_C += actual_c;
611
612   return 0;
613 }
614
615 int go_len_block(double a, double b, double c, int f_speed) {
616
617   //calculate number of steps and direction of movement to move
       from current pos to new pos.
618   int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, a);
619   int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, b);
620   int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, c);
621
622
623   //Setting important info in stepper classes
624   stepperA.setCurrentPosition(0);
625   stepperB.setCurrentPosition(0);
626   stepperC.setCurrentPosition(0);
627   int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
628   stepperA.setMaxSpeed(max_speed);
629   stepperB.setMaxSpeed(max_speed);
630   stepperC.setMaxSpeed(max_speed);
631   //move the payload to location
632   long steps[] = {steps_A, -steps_B, steps_C};
633
634   steppers.moveTo(steps);
635
636   //blocks until steppers finish moving
637   while(steppers.run()){
638   }
639
640   //find actual displacement
641   double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
642   double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP)
       ;
643   double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;
644
645   //after done, update current values
646   CURRENT_THREAD_LEN_A += actual_a;
647   CURRENT_THREAD_LEN_B += actual_b;
648   CURRENT_THREAD_LEN_C += actual_c;
649
650   X_ADDRESS_TARGET = 0;
651   Y_ADDRESS_TARGET = 0;
652   Z_ADDRESS_TARGET = 0;
653
654   return 0;
```

```
655 }
656
657
658 int go(double x, double y, double z, int f_speed) {
659   X_ADDRESS_TARGET = x;
660   Y_ADDRESS_TARGET = y;
661   Z_ADDRESS_TARGET = z;
662   F_PERCENT_SPEED = f_speed;
663
664   //calculate new necessary thread lengths
665   double new_len_a = calculate_thread_length_A_h(x,y,z);
666   double new_len_b = calculate_thread_length_B_h(x,y,z);
667   double new_len_c = calculate_thread_length_C_h(x,y,z);
668
669   /*Serial.println("");
670   Serial.print("New Length A: ");
671   Serial.println(new_len_a);
672   Serial.print("New Length B: ");
673   Serial.println(new_len_b);
674   Serial.print("New Length C: ");
675   Serial.println(new_len_c);
676
677   Serial.println("");
678   Serial.print("CurrentLength A: ");
679   Serial.println(CURRENT_THREAD_LEN_A);
680   Serial.print("CurrentLength B: ");
681   Serial.println(CURRENT_THREAD_LEN_B);
682   Serial.print("CurrentLength C: ");
683   Serial.println(CURRENT_THREAD_LEN_C);*/
684
685   //calculate number of steps and direction of movement to move
       from current pos to new pos.
686   int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A,
       new_len_a);
687   int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B,
       new_len_b);
688   int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C,
       new_len_c);
689
690
691   //Setting important info in stepper classes
692   stepperA.setCurrentPosition(0);
693   stepperB.setCurrentPosition(0);
694   stepperC.setCurrentPosition(0);
695   int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
696   stepperA.setMaxSpeed(max_speed);
697   stepperB.setMaxSpeed(max_speed);
698   stepperC.setMaxSpeed(max_speed);
699   //stepperA.setAcceleration(max_speed/5);
700   //stepperB.setAcceleration(max_speed/5);
701   //stepperC.setAcceleration(max_speed/5);
702   //move the payload to location
703   long steps[] = {steps_A, -steps_B, steps_C};
704
705   lcd.clear();
706   lcd.print("Moving to: F");
707   lcd.setCursor(0, 1);
708   lcd.print(x);
```

```
709   lcd.print(" ");
710   lcd.print(y);
711   lcd.print(" ");
712   lcd.print(z);
713
714   steppers.moveTo(steps);
715
716   //move_to(steps_A, steps_B, steps_C);
717
718   ON_MOVE = true;
719
720   //blocks until steppers finish moving
721   while(steppers.run() && ON_MOVE){
722     Commands.available();
723   }
724
725   //find actual displacement
726   double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
727   double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP)
       ;
728   double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;
729
730   /*Serial.println("");
731   Serial.print("DELTA A:");
732   Serial.println(actual_a);
733   Serial.print("DELTA B:");
734   Serial.println(actual_b);
735   Serial.print("DELTA C:");
736   Serial.println(actual_c);
737
738   Serial.println("\nFinal Stepper Pos:");
739   Serial.print("Stepper A: ");
740   Serial.println(stepperA.currentPosition());
741   Serial.print("Stepper B: ");
742   Serial.println(-stepperB.currentPosition());
743   Serial.print("Stepper C: ");
744   Serial.println(stepperC.currentPosition());*/
745
746   ON_MOVE = false;
747
748   //after done, update current values
749   if (!M2_EN && !M6_EN){
750     CURRENT_X = x;
751     CURRENT_Y = y;
752     CURRENT_PAYLOAD_HEIGHT = z;
753   }
754   if (!M6_EN){
755     CURRENT_THREAD_LEN_A += actual_a;
756     CURRENT_THREAD_LEN_B += actual_b;
757     CURRENT_THREAD_LEN_C += actual_c;
758   }
759   M2_EN = false;
760
761   X_ADDRESS_TARGET = 0;
762   Y_ADDRESS_TARGET = 0;
763   Z_ADDRESS_TARGET = 0;
764
765   /*Serial.println("");
```

```
766   Serial.print("Final Length A: ");
767   Serial.println(CURRENT_THREAD_LEN_A);
768   Serial.print("Final Length B: ");
769   Serial.println(CURRENT_THREAD_LEN_B);
770   Serial.print("Final Length C: ");
771   Serial.println(CURRENT_THREAD_LEN_C);*/
772
773   return 0;
774 }
775
776 ////////////////////////CALCULATIONS FOR MOVEMENT
      ////////////////////////
777
778 void init_values(){ //expects in not mm
779   lcd.clear();
780   lcd.print("Location:");
781   lcd.setCursor(0, 1);
782   lcd.print("Z, X, Y");
783   while(Serial.available() <= 0){
784   }
785   double h = Serial.parseFloat();
786   double x = Serial.parseFloat();
787   double y = Serial.parseFloat();
788
789   CURRENT_PAYLOAD_HEIGHT = h;
790
791   double La = calculate_thread_length_A_h(x,y,h);
792   double Lb = calculate_thread_length_B_h(x,y,h);
793   double Lc = calculate_thread_length_C_h(x,y,h);
794
795   CURRENT_THREAD_LEN_A = La;
796   CURRENT_THREAD_LEN_B = Lb;
797   CURRENT_THREAD_LEN_C = Lc;
798   CURRENT_X = x;
799   CURRENT_Y = y;
800
801   lcd.clear();
802   lcd.print("Z: ");
803   lcd.print(CURRENT_PAYLOAD_HEIGHT);
804   lcd.setCursor(0, 1); //2nd Line 1st char
805   lcd.print("X: ");
806   lcd.print(CURRENT_X);
807   lcd.setCursor(8, 1); //2nd Line 9th char
808   lcd.print("Y: ");
809   lcd.print(CURRENT_Y);
810
811 }
812
813 void ready_command_lcd(){
814   lcd.clear();
815   lcd.setCursor(0, 0); //2nd Line 1st char
816   lcd.print("    <READY>    ");
817   lcd.setCursor(0, 1); //2nd Line 1st char
818   lcd.print("   Listening   ");
819 }
820
821 void send_sensor(){
822
```

```
823
824   sensorVal = analogRead(sensor);
825   lightLevel = (sensorVal/1023)*10;
826   Serial.write("L");
827   Serial.print(lightLevel);
828
829   boolean success = false;
830   uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the
       returned UID
831   uint8_t uidLength;                        // Length of the UID (4
       or 7 bytes depending on ISO14443A card type)
832   success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid
       [0], &uidLength);
833   long test = 0;
834   if (success){
835     for (uint8_t i=0; i < uidLength; i++)
836     {
837       //Serial.print(" 0x");Serial.print(uid[i], HEX);
838       test = test + uid[i];
839       if(i<3){
840       test = test * 256;
841       }
842     }
843     Serial.write("R");
844     Serial.write(test);
845   } else {
846     Serial.write("R0");
847   }
848 }
849
850 void task_done(){
851   X_ADDRESS_TARGET = 0;
852   Y_ADDRESS_TARGET = 0;
853   Z_ADDRESS_TARGET = 0;
854   F_PERCENT_SPEED = 100;
855   OPCODE = 1000; //change OPCODE to 1000 when task is done
856   //delay(2000);
857   ready_command_lcd();
858
859   Serial.write("X");
860   Serial.print(CURRENT_X);
861   Serial.write("Y");
862   Serial.print(CURRENT_Y);
863   Serial.write("Z");
864   Serial.print(CURRENT_PAYLOAD_HEIGHT);
865   Serial.write("A");
866   Serial.print(CURRENT_THREAD_LEN_A);
867   Serial.write("B");
868   Serial.print(CURRENT_THREAD_LEN_B);
869   Serial.write("C");
870   Serial.print(CURRENT_THREAD_LEN_C);
871
872   send_sensor();
873
874   delay(1000);
875
876   Serial.write("G"); //let matlab know when it is good to send
       another command.
```

```
877
878 }
879
880 void set_pins(){
881   //LCD
882   pinMode(2, OUTPUT);
883   pinMode(3, OUTPUT);
884   pinMode(4, OUTPUT);
885   pinMode(5, OUTPUT);
886   pinMode(8, OUTPUT);
887
888   //STEPPERS
889   pinMode(25, OUTPUT);
890   pinMode(23, OUTPUT);
891   pinMode(29, OUTPUT);
892   pinMode(27, OUTPUT);
893   pinMode(33, OUTPUT);
894   pinMode(31, OUTPUT);
895
896   pinMode(35, OUTPUT);
897   pinMode(37, OUTPUT);
898   pinMode(39, OUTPUT);
899
900 }
901
902 void setup() {
903   Commands.begin();
904   lcd.begin(16, 2);
905   lcd.print("      <FP>      ");
906   lcd.setCursor(2, 1);
907   lcd.print("Spydercam 18");
908   delay(2000);
909   lcd.clear();
910   lcd.print("      BETA      ");
911   lcd.setCursor(0, 1);
912   lcd.print("Arduino Firmware");
913   delay(2000);
914   init_values();
915   delay(2000);
916   set_pins();
917   stepperA = AccelStepper(motorInterfaceType, stepPinA, dirPinA);
918   stepperB = AccelStepper(motorInterfaceType, stepPinB, dirPinB);
919   stepperC = AccelStepper(motorInterfaceType, stepPinC, dirPinC);
920
921   digitalWrite(35, LOW);
922   digitalWrite(37, LOW);
923   digitalWrite(39, LOW);
924
925   steppers.addStepper(stepperA);
926   steppers.addStepper(stepperB);
927   steppers.addStepper(stepperC);
928
929
930   //sensor setup
931   nfc.begin();
932   lcd.clear();
933   lcd.print("  Enabling NFC  ");
934   lcd.setCursor(0, 1);
```

```
  lcd.print(" In Progress... ");
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    lcd.clear();
    lcd.print("     ERROR      ");
    lcd.setCursor(0, 1);
    lcd.print("   NFC Failed   ");
    while (1); // Halt
  }

  delay(1000);
  nfc.SAMConfig();
  ready_command_lcd();

  //Serial.write("G");
}

void loop() {
  if (Commands.available()) {
    task_done();
  }
}
```

8.2 MATLAB Code

```
% ======================================== %
% === MATLAB GUI FOR SPYDERCAM TEAM 18 === %
% ======================================== %

% === Main Program === %
clear;

% --- Variables --- %
global drawing_input; %Cell array holding sets of user drawn lines
drawing_input = cell(1);

global num_sets;   %Number of user entered sets of lines. (Sets are
    connected lines
num_sets = 0;

global home;       %Home Location for Payload
home = [0 0 3];

global draw_height; %Height (Z Pos) for Drawing with Pen attachment
draw_height = 3.6;

global non_draw_height; %Height (Z Pos) for NOT Drawing with pen
non_draw_height = 4.6;

global gcode_buffer; %Buffer for holding G-Code to be sent to
   Arduino
gcode_buffer = [];

global sensor_data; %Incoming sensor data from the Arduino is
   stored here
sensor_data = zeros(0,5);

global serial_on;   %Turn on or off incoming serial (1 = on, 0 =
   off)
serial_on = 1;

global prgm_end;    %Set to 0 if the current Gcode sequence is over
prgm_end = 1;

% --- Serial --- %
global s;   %Serial port. Open it up. Throw an error and quit if it
    wont open
try s = serialport("COM3",9600);
catch
    disp("Serial device not connected. Reconnect and restart");
    return;
end
configureCallback(s,"byte",1,@readSerialData) %Callback function
   for serial. Executes if there is 1 byte available

% --- GUI Figure --- %
global gui; %GUI uifigure and overarching labels
gui = uifigure(1,'Position', [250, 70, 1100, 700],'Name','SpyderCam
    Team 18 MATLAB GUI');

% Main area labels
```

```
drw_lbl = uilabel(gui,'Text', 'Drawing Input', 'Position',[45 660 200 40],'FontSize',24);
gcode_lbl = uilabel(gui,'Text', 'G-Code Input', 'Position',[400 660 200 40],'FontSize',24);
ea_lbl = uilabel(gui,'Text', 'Execution Area', 'Position',[740 660 200 40],'FontSize',24);

% Child figures
global sensorViewFig; %Figure that shows sensor data
global bufferfig;   %Figure that shows Gcode buffer

% --- User Drawing Input Area --- %
draw_area = uiaxes(gui,'Position',  [20, 227, 340, 440],'ButtonDownFcn',@axesCallback); %Create Axes
draw_area.XLim = [0 8.5]; %Set axes limits
draw_area.YLim = [0 11];
draw_area.PickableParts = 'all'; %Make axes clickable
draw_area.HitTest = 'off'; %Make axes not clickable
draw_area.NextPlot = 'replacechildren'; %Each plot replaces the last, but axes settings stay the same

% Buttons for adding a new line set or clearing the drawing
draw_line = uibutton(gui,'state','Text','Draw Line','Position',[30, 197, 100, 22],'ValueChangedFcn',{@drawBtnCallback,draw_area}); %State button: pushed in is drawing mode, pushed out = finished drawing line set
clear_drawing = uibutton(gui,'push','Text','Clear Drawing','Position',[140, 197, 100, 22],'ButtonPushedFcn',{@clearBtnCallback,draw_area}); %Push button: clears everything in drawing window
slbl = uilabel(gui,'Text','Speed (IPM)','Position',[25, 167, 70, 25]); %Label for Speed Slider
speed = uislider(gui,'Position',[110, 180, 230, 3],'Limits',[0.1 4.5]);  %Speed Slider
convert_to_gcode = uibutton(gui,'push','Text','Convert to G-Code ->','Position',[250, 197, 110, 22]); %Convert to G-Code Button


% --- GCODE Input Area --- %
gcode_area = uitextarea(gui,'Position',[400, 270, 300, 390]); %Text box for inputting GCODE
gcode_area.Value = sprintf(";"); %Initialize gcode text input area
gcode_dropdown = uidropdown(gui,'Items',{'G00','G01','G04','G20','G21','G28','G90','G91','M00','M02','M06','M30','M72'},'Position',[400,230,100,22],'ValueChangedFcn',@GdownCallback); %Dropdown for choosing G-code
arg_boxes(); %Set up boxes for inputting arguments
disp_args('G00','000'); %Initialize argument boxes with the ones for G00
global currentgcode; %Current G-Code being formulated with dropdown and argument input boxes
enter_gcode = uibutton(gui,'push','Text','Enter G-Code','Position',[400, 190, 100, 22],'ButtonPushedFcn',{@enterCallback,gcode_area}); %Send current G-Code into the G-Code text area
line_break = uibutton(gui,'push','Text','Line Break','Position',[520, 190, 100, 22],'ButtonPushedFcn',{@linebreakCallback,gcode_area}); %Add a line break (;)
delete_line = uibutton(gui,'push','Text','Delete Line','Position'
```

```
    ,[400, 150, 100, 22],'ButtonPushedFcn',{@deleteLineCallback,
    gcode_area});%Delete 1 line from the G-Code text area
84 clear_gcode = uibutton(gui,'push','Text','Clear G-Code','Position'
    ,[520, 150, 100, 22],'ButtonPushedFcn',{@clearGcodeCallback,
    gcode_area}); %Clear all G-Code from text area
85
86 convert_to_gcode.ButtonPushedFcn = {@convertCallback, gcode_area,
    speed}; %Set callback for convert_to_gcode button. Down here so
    it can reference speed
87
88
89 % --- Execution Area --- %
90 incr_distance = uieditfield(gui,'Position',[800, 270, 50, 20]); %
    Incremental movement distance
91 incr_dlbl = uilabel(gui,'Text','Distance','Position',[750, 270, 60,
     25]); %label for ^
92 incr_speed = uieditfield(gui,'Position',[940, 270, 50,20]); %
    Incremental movement speed
93 incr_slbl = uilabel(gui,'Text','Speed (IPM)','Position',[865, 270,
    70,25]); %Label for ^
94
95 % X, Y, and Z incremental movement buttons
96 yplus_btn = uibutton(gui,'push','Text','Y+','Position',[850, 530,
    100, 100],'ButtonPushedFcn',{@directionBtnCallback,"Y+",
    incr_distance,incr_speed});
97 xminus_btn = uibutton(gui,'push','Text','X-','Position',[740, 420,
    100, 100],'ButtonPushedFcn',{@directionBtnCallback,"X-",
    incr_distance,incr_speed});
98 xplus_btn = uibutton(gui,'push','Text','X+','Position',[960, 420,
    100, 100],'ButtonPushedFcn',{@directionBtnCallback,"X+",
    incr_distance,incr_speed});
99 yminus_btn = uibutton(gui,'push','Text','Y-','Position',[850, 310,
    100, 100],'ButtonPushedFcn',{@directionBtnCallback,"Y-",
    incr_distance,incr_speed});
100 zplus_btn = uibutton(gui,'push','Text','Z+','Position',[960, 530,
    50, 50],'ButtonPushedFcn',{@directionBtnCallback,"Z+",
    incr_distance,incr_speed});
101 zminus_btn = uibutton(gui,'push','Text','Z-','Position',[790, 360,
    50, 50],'ButtonPushedFcn',{@directionBtnCallback,"Z-",
    incr_distance,incr_speed});
102 home_btn = uibutton(gui,'push','Text','Home','Position',[850, 420,
    100, 100],'ButtonPushedFcn',{@directionBtnCallback,"home",
    incr_distance,incr_speed}); %Button for sending the payload to
    the home locations
103
104 % Buffer and sensor data control buttons
105 send_btn = uibutton(gui,'push','Text','Send G-Code to Buffer (From
    Text Area)','Position',[740, 150, 270, 50],'ButtonPushedFcn',{
    @send2bufferCallback,gcode_area}); %Sends G-Code from text area
    to buffer
106 run_btn = uibutton(gui,'state','Text','Run','Position',[1020, 150,
    40, 50],'ValueChangedFcn',@runCallback,'Tag','RUN'); %Start the
    gcode sequence
107 sensor_data_btn = uibutton(gui,'push','Text', 'Sensor Data','
    Position',[740, 210, 100, 20],'ButtonPushedFcn',
    @sensorViewCallback); %Show the sensor data window
108 buffer_btn = uibutton(gui,'push','Text', 'G-Code Buffer','Position'
    ,[845, 210, 100, 20],'ButtonPushedFcn',@bufferCallback); %
```

```
    Displays the contents of the buffer in a new window
clear_buffer_btn = uibutton(gui,'push','Text','Clear Buffer (0)','
    Position',[950, 210, 110, 20],'Tag','C','ButtonPushedFcn',
    @bufferClearCallback); %Clears the contents of the buffer (if
    they haven't been sent to the Arduino)

% Direct Serial Area
ds_lbl = uilabel(gui,'Text','Direct Serial Comm','Position',[40,
    100, 300, 40],'FontSize',24); % Label for direct serial area
ser_get = uitextarea(gui,'Position',[30 50 950, 50],'Tag','SER','
    ValueChangedFcn',@serGetCallback); % Box for receiving serial
dir_ser = uitextarea(gui,'Position',[30 20 950, 20]); % Box for
    sending serial
dir_send = uibutton(gui,'push','Text','Send','Position',[990 20 70,
     80],'ButtonPushedFcn',{@dirSendCallback,dir_ser,ser_get}); %
    Button that sends serial

% === UTILITY FUNCTIONS === %
function plot_input(obj,new_point)
    % This is the plotting function. It takes in the axes object
    and a new
    % point. It adds the point to drawing input, then plots all of
    drawing
    % input. Inputing [-1 -1] clears the plot

    global drawing_input;
    global num_sets;

    if new_point ~= [-1 -1] %Only plot if the new point is not [-1
    -1];
        drawing_input{num_sets+1} = [drawing_input{num_sets+1};
    new_point]; %Add new point to drawing input

        inplot = cell(num_sets+1); %Input plot
        obj.NextPlot = 'add'; % Change plot settings so that plots
    get added to existing plots
        for i = 1:1:num_sets+1 % Draw each cell (1 cell = 1 line
    set) of drawing input. Make them not clickable.
            inplot{i} = plot(obj,drawing_input{i}(:,1),
    drawing_input{i}(:,2),'r-o');
            inplot{i}.HitTest = 'off';
        end
        obj.NextPlot = 'replacechildren'; %Change plot back to
    replaceable
    else %If new_point is [-1 -1]
        plot(obj,-1,-1); %Just plot 1 garbage point off the axes
    end
end

function disp_args(code,prevcode)
    % This is the display arguments function. This function turns
    on or off
    % the visibility of each argument editfield depending on which
    G-Code
    % has been chosen in the dropdown menu. code and prevcode refer
    to the
    % new and old values of the dropdown, respectively.

```

```
147     global gui;
148     global currentgcode;
149     children = get(gui,'Children');
150 
151     % If dropdown changed from G00
152     if sum(prevcode == 'G00') == 3
153         %Find the editfields/labels with the G00 tag and make them
    invisible.
154         for i = 1:1:length(children)
155             if ~isempty(children(i).Tag) && (children(i).Tag == "
    G00")
156                 children(i).Visible = 'off';
157                 %Find the editfield and set its value to empty
158                 if length(children(i).Type) == 11
159                     children(i).Value = '';
160                 end
161             end
162         end
163     %Find the editfields/labels with the G01 or G00 tag and make
    them
164     %invisible. (G00 and G01 both have XYZ, G01 just also has F)
165     elseif sum(prevcode == 'G01') == 3
166         for i = 1:1:length(children)
167             if ~isempty(children(i).Tag) && ((children(i).Tag == "
    G00") || (children(i).Tag == "G01"))
168                 children(i).Visible = 'off';
169                 %Find the editfield and set its value to empty
170                 if length(children(i).Type) == 11
171                     children(i).Value = '';
172                 end
173             end
174         end
175     %Find the editfields/labels with the G04 tag and make them
    invisible
176     elseif sum(prevcode == 'G04') == 3
177         for i = 1:1:length(children)
178             if ~isempty(children(i).Tag) && (children(i).Tag == "
    G04")
179                 children(i).Visible = 'off';
180                 %Find the editfield and set its value to empty
181                 if length(children(i).Type) == 11
182                     children(i).Value = '';
183                 end
184             end
185         end
186     else
187 
188     end
189 
190     %If the new code is G00, turn editfields/labels tagged G00
    visible
191     if sum(code == 'G00') == 3
192         for i = 1:1:length(children)
193             if ~isempty(children(i).Tag) && (children(i).Tag == "
    G00")
194                 children(i).Visible = 'on';
195             end
196         end
```

```
197         %Set the current G-Code to be filled with arguments
198         currentgcode = ["G00" "" "" ""];
199     %If the new code is G01, turn editfields/labels tagged G00 or
    G01 visible
200     elseif sum(code == 'G01') == 3
201         for i = 1:1:length(children)
202             if ~isempty(children(i).Tag) && ((children(i).Tag == "
    G00") || (children(i).Tag == "G01"))
203                 children(i).Visible = 'on';
204             end
205         end
206         %Set the current G-Code to be filled with arguments
207         currentgcode = ["G01" "" "" "" ""];
208     %If the new code is G04, turn editfields/labels tagged G04
    visible
209     elseif sum(code == 'G04') == 3
210         for i = 1:1:length(children)
211             if ~isempty(children(i).Tag) && (children(i).Tag == "
    G04")
212                 children(i).Visible = 'on';
213             end
214         end
215         %Set the current G-Code to be filled with arguments
216         currentgcode = ["G04" ""];
217     else
218         %Set the current G-Code (no arguments)
219         currentgcode = [convertCharsToStrings(code)];
220     end
221 end
222
223 function arg_boxes()
224     % This function sets up the editfields and labels for argument
    input on
225     % the G-Code Input. It puts all of the items in position, then
    makes
226     % them all invisible
227
228     global gui;
229
230     % XYZ labels/editfields
231     xl = uilabel(gui,'Text', 'X','Position',[520 230 10 20],'Tag',"
    G00");
232     xe = uieditfield(gui,'Position',[535 230 30 20],'Tag',"G00",'
    ValueChangedFcn',{@fieldChange,'X','0'});
233     yl = uilabel(gui,'Text', 'Y','Position',[580 230 10 20],'Tag',"
    G00");
234     ye = uieditfield(gui,'Position',[595 230 30 20],'Tag',"G00",'
    ValueChangedFcn',{@fieldChange,'Y','0'});
235     zl = uilabel(gui,'Text', 'Z','Position',[640 230 10 20],'Tag',"
    G00");
236     ze = uieditfield(gui,'Position',[655 230 30 20],'Tag',"G00",'
    ValueChangedFcn',{@fieldChange,'Z','0'});
237
238     % F label/editfield
239     fl = uilabel(gui,'Text', 'F','Position',[640 190 10 20],'Tag',"
    G01");
240     fe = uieditfield(gui,'Position',[655 190 30 20],'Tag',"G01",'
    ValueChangedFcn',{@fieldChange,'F','0'});
```

```
241
242     % PX dropdown/editfield
243     pxdd = uidropdown(gui,'Items',{'P','X'},'Position',[520 230 40
    20],'Tag',"G04");
244     pxe = uieditfield(gui,'Position',[565 230 30 20],'Tag',"G04",'
    ValueChangedFcn',{@fieldChange,'P',pxdd});
245     pxdd.ValueChangedFcn = {@pxChange,pxe};
246
247     % Turn all of the elements created in this function invisible
248     xl.Visible = 'off';
249     xe.Visible = 'off';
250     yl.Visible = 'off';
251     ye.Visible = 'off';
252     zl.Visible = 'off';
253     ze.Visible = 'off';
254     fl.Visible = 'off';
255     fe.Visible = 'off';
256     pxdd.Visible = 'off';
257     pxe.Visible = 'off';
258
259 end
260
261 function [gcode] = drawing2gcode(speed)
262     % This function turns data from the user inputted drawing and
    turns it
263     % into G-Code. It takes in a constant speed for the whole
    operation. It
264     % orders the drawing lines by proximity and then turns that
    path into
265     % G-Code
266
267     %Setup
268     F = speed;
269     global home;
270     global drawing_input;
271     sets = length(drawing_input);
272     if length(drawing_input{sets}) < 1
273        sets = sets-1;
274     end
275     indexes = [];
276
277     %Find the line set that starts closest to the origin
278     lowest_dist = 100;
279     for i = 1:1:sets
280        temp = sqrt(sum((home(1:2) - drawing_input{i}(1,:)).^2));
281        if temp < lowest_dist
282             lowest_dist = temp;
283             indexes(1) = i;
284        end
285     end
286
287     %Add line set that starts closest to the origin to a new
    ordered array
288     new_order{1} = drawing_input{indexes(1)};
289
290     % Order the rest of of the line sets, next closest starting
    point after
291     % current set endpoint.
```

```
292     while length(new_order) < sets
293     lowest_dist = 100;
294         for i = 1:1:sets
295             if ~ismember(i,indexes)
296                 temp = sqrt(sum((new_order{end}(end,:) -
    drawing_input{i}(1,:)).^2));
297                 if temp < lowest_dist
298                     lowest_dist = temp;
299                     indexes(length(indexes)+1) = i;
300                     new_order{length(new_order)+1} = drawing_input{
    i};
301                 end
302             end
303         end
304     end
305 
306     %Initialize gcode
307     gcode = {};
308     gcode{1} = sprintf(';');
309 
310     % Turn each set into a set of G-Code Commands
311     for i = 1:1:sets
312         %Go to location of first point and lower down into pen
    height
313         gcode{length(gcode)+1} = ['G00' ' X' num2str(new_order{i
    }(1,1)) ' Y' num2str(new_order{i}(1,2)) ' Z3;'];
314         gcode{length(gcode)+1} = ['G01' ' Z2' ' F' num2str(F) ';'];
315         %Stay in pen heigh and go through the points in the set (
    draw the
316         %line)
317         for j = 2:1:length(new_order{i})
318             gcode{length(gcode)+1} = ['G01' ' X' num2str(new_order{
    i}(j,1)) ' Y' num2str(new_order{i}(j,2)) ' F' num2str(F) ';'];
319         end
320         %Lift up out of pen height
321         gcode{length(gcode)+1} = ['G01' ' Z3' ' F' num2str(F) ';'];
322     end
323 
324 end
325 % ====================================== %
326 
327 
328 
329 % === GUI ELEMENT CALLBACK FUNCTIONS === %
330 function axesCallback(obj,~)
331     % This is the callback function for clicking on the Drawing
    Axes.
332     % It gets the mouse click location and sends it to the plotting
333     % function.
334 
335     mouse_loc = obj.CurrentPoint;
336     plot_input(obj,mouse_loc(1,1:2));
337 end
338 
339 function drawBtnCallback(obj,~,obj2)
340     % This is the callback function for clicking the "Draw Line"
    Button.
341     % It turns on the ability to draw on the axes.
```

```
342
343     global drawing_input;
344     global num_sets;
345
346     if obj.Value == 1                       % If button is pressed
347         obj2.HitTest = 'on';                    % Turn axis clicking on
348         obj.Text = 'End Line';
349     else                                    % If button is not pressed
    (un-pressed? pulled?)
350         obj2.HitTest = 'off';                   % Turn axis clicking
    off
351         obj.Text = 'Draw Line';
352         if ~isempty(drawing_input{num_sets+1}) % If something has
    been drawn in this set
353             num_sets = num_sets + 1;            % Add 1 to num_sets
354             drawing_input{num_sets+1} = [];         % Add another
    cell to user input
355         end
356     end
357 end
358
359 function clearBtnCallback(~,~,obj2)
360     % This is the callback function for the "Clear Drawing" button.
361     % It just resets the user input/number of sets then plots an
    empty
362     % graph.
363
364     global drawing_input;
365     global num_sets;
366
367     num_sets = 0;
368     drawing_input = cell(1);
369
370     plot_input(obj2,[-1 -1]);
371
372 end
373
374 function GdownCallback(obj,event)
375     %This is the callback function for the G-Code dropdown menu. It
     grabs
376     %the current and previous values, then sends them to the
    disp_args
377     %function
378
379     val = obj.Value;
380     prevVal = event.PreviousValue;
381     disp_args(val,prevVal);
382
383 end
384
385 function enterCallback(~,~,text_area)
386     %This is the callback function for the send gcode button. It
    adds the
387     %G-Code formed by the dropdown and argument inputs to the text
    area.
388
389     global currentgcode;
390     index = length(text_area.Value)+1;
```

```
391     text_area.Value{index} = sprintf('');
392     for i = 1:1:length(currentgcode)
393         charcode = convertStringsToChars(currentgcode(i));
394         if ~isempty(charcode)
395             text_area.Value{index} = [text_area.Value{index}
    sprintf(charcode)  sprintf(' ')];
396         end
397     end
398     text_area.Value{index} = text_area.Value{index}(1:end-1);
399     text_area.Value{index} = [text_area.Value{index} sprintf(';')];
400 
401 end
402 
403 function fieldChange(obj,~,lbl,obj2)
404     %This is the callback function for when the argument fields are
405     %updated. This function runs every time those are changed
    values.
406 
407     global currentgcode;
408 
409     %Depending on the argument label, add argument to current g
    code
410     if lbl == 'X'
411         currentgcode(2) = convertCharsToStrings([lbl obj.Value]);
412     elseif lbl == 'Y'
413         currentgcode(3) = convertCharsToStrings([lbl obj.Value]);
414     elseif lbl == 'Z'
415         currentgcode(4) = convertCharsToStrings([lbl obj.Value]);
416     elseif lbl == 'F'
417         currentgcode(5) = convertCharsToStrings([lbl obj.Value]);
418     elseif lbl == 'P'
419         currentgcode(2) = convertCharsToStrings([obj2.Value obj.
    Value]);
420     else
421 
422     end
423 
424 end
425 
426 function convertCallback(~,~,text_area,speed)
427     % This is the callback function for the convert to G-Code
    button. It
428     % calls the drawing2gcode function on the drawing_input, then
    puts the
429     % results in the text area.
430 
431     global drawing_input;
432     if length(drawing_input{1}) > 0
433         gcode = drawing2gcode(speed.Value);
434         text_area.Value = gcode;
435     end
436 
437 end
438 
439 function linebreakCallback(~,~,text_area)
440     %This is the callback function for the line break button. It
    just
441     %prints a ';' on the next line.
```

```
442
443     index = length(text_area.Value)+1;
444     text_area.Value{index} = sprintf(';');
445
446 end
447
448 function deleteLineCallback(~,~,text_area)
449     %This is the callback function for the delete line button. It
        just
450     %deletes the last line in the G-Code text box
451
452     temp = {};
453     if length(text_area.Value) > 1
454         for i = 1:1:(length(text_area.Value)-1)
455             temp{i} = text_area.Value{i};
456         end
457         text_area.Value = temp;
458     end
459
460 end
461
462 function pxChange(obj,~,ef)
463     % This is the callback for the P and X dropdown on the G04
        argument
464     % inputs. It updates the currentgcode value and also deletes
        the
465     % editfield value if the drop down changes value
466
467     global gui;
468     global currentgcode;
469     children = get(gui,'Children');
470
471     for i = 1:1:length(children)
472        if length(children(i).Type) == 11
473             children(i).Value = '';
474        end
475     end
476     currentgcode(2) = convertCharsToStrings([obj.Value ef.Value]);
477
478 end
479
480 function clearGcodeCallback(~,~,text_area)
481     % This is the callback function for the clear gcode button. It
        just
482     % sets the text_area to be equal ';'
483
484     text_area.Value = sprintf(';');
485
486 end
487
488 function send2bufferCallback(~,~,text_area)
489     % This function is the callback for the send to buffer button.
        It adds
490     % the lines from the text area to the buffer, then deletes the
        text
491     % area.
492
493     global gcode_buffer;
```

```
494     for i = 1:1:length(text_area.Value)
495         gcode_buffer = [{text_area.Value{i}}; gcode_buffer];
496     end
497     text_area.Value = {''};
498     update_clear_btn();
499     update_bufferfig();
500 
501 end
502 
503 function directionBtnCallback(~,~,dir,dist,speed)
504     % This is the callback function for the directional buttons. It
        takes a
505     % direction, distance, and speed, and then ti moves at that
        speed for
506     % that dist in that direction.
507 
508     global gcode_buffer;
509 
510    %If user didn't set distance or speed, give them some default
        values
511    if isempty(dist.Value)
512        dist.Value = '1';
513    end
514    if isempty(speed.Value)
515        speed.Value = '100';
516    end
517 
518    % If direction = Y+,Y-,X+,X-,Z+,Z-
519    %    add 'G91' to buffer (Set incremental movement)
520    %    add 'G01' to buffer with distance and speed (Move in the
        specified
521    %        direction
522    %    add 'G90' to buffer (Set absolute movement)
523    if dir == "Y+"
524        gcode_buffer = [{'G91;'}; gcode_buffer ];
525        gcode_buffer = [{['G01 Y'  dist.Value  ' F'  speed.Value  ';
        ']}; gcode_buffer ];
526        gcode_buffer = [{'G90;'}; gcode_buffer ];
527    elseif dir == "X-"
528        gcode_buffer = [{'G91;'}; gcode_buffer ];
529        gcode_buffer = [{['G01 X-'  dist.Value  ' F'  speed.Value  '
        ;']}; gcode_buffer ];
530        gcode_buffer = [{'G90;'}; gcode_buffer ];
531    elseif dir == "X+"
532        gcode_buffer = [{'G91;'}; gcode_buffer ];
533        gcode_buffer = [{['G01 X'  dist.Value  ' F'  speed.Value  ';
        ']}; gcode_buffer ];
534        gcode_buffer = [{'G90;'}; gcode_buffer ];
535    elseif dir == "Y-"
536        gcode_buffer = [{'G91;'}; gcode_buffer ];
537        gcode_buffer = [{['G01 Y-'  dist.Value  ' F'  speed.Value  '
        ;']}; gcode_buffer ];
538        gcode_buffer = [{'G90;'}; gcode_buffer ];
539    elseif dir == "Z+"
540        gcode_buffer = [{'G91;'}; gcode_buffer ];
541        gcode_buffer = [{['G01 Z'  dist.Value  ' F'  speed.Value  ';
        ']}; gcode_buffer ];
542        gcode_buffer = [{'G90;'}; gcode_buffer ];
```

```
543     elseif dir == "Z-"
544         gcode_buffer = [{'G91;'}; gcode_buffer ];
545         gcode_buffer = [{['G01 Z-'  dist.Value  ' F'  speed.Value  '
    ;']}; gcode_buffer ];
546         gcode_buffer = [{'G90;'}; gcode_buffer ];
547     elseif dir == "home"
548         gcode_buffer = [{'G28;'}; gcode_buffer ];
549     else
550 
551     end
552 
553     %Update the clear button because values have been added to the
    buffer
554     update_clear_btn();
555     update_bufferfig();
556 
557 end
558 
559 function bufferCallback(~,~)
560     %This is the view gcode buffer button callback. It shows a
    figure which
561     %contains a text area which contains the contents of the G-Code
     Buffer
562     global bufferfig;
563     global gcode_buffer;
564     bufferfig = uitextarea('Position',[50 20 460, 380]);
565     bufferfig.Parent.Name = 'G-Code Buffer';
566 
567     bufferfig.Value = flip(gcode_buffer);
568 
569 end
570 
571 function bufferClearCallback(~,~)
572     %This function is the buffer clear button callback. It clears
    the
573     %buffer and updates the number on itself.
574 
575     global gcode_buffer;
576     gcode_buffer = [];
577     update_clear_btn();
578     update_bufferfig();
579 
580 end
581 
582 function sensorViewCallback(~,~)
583     %This is the callback function for the button that opens the
    sensor
584     %data view window. It opens the sensor data view window.
585 
586     global sensorViewFig;
587 
588     %Make the figure and the headings
589     sensorViewFig = uifigure('Position',[200 100 600 500],'Tag','
    svf');
590     axes_label = uilabel(sensorViewFig,'Text', 'Data Points', '
    Position',[45 460 200 40],'FontSize',24);
591     info_label = uilabel(sensorViewFig,'Text', 'Sensor Data', '
    Position',[390 460 200 40],'FontSize',24);
```

```
592
593     %Label and values for coordinates (data from Arduino)
594     coord_label = uilabel(sensorViewFig,'Text', 'Coordinates (
    Inches):', 'Position',[390 430 200 25],'FontSize',16);
595     coord_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390
    410 200 25],'Tag','COORD');
596
597     %Label and value for light sensor (data from Arduino)
598     lsv_label = uilabel(sensorViewFig,'Text', 'Light Sensor Value:'
    , 'Position',[390 360 200 25],'FontSize',16);
599     lsv_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390
    340 200 25],'Tag','LSV');
600
601     %Label and value for RFID (data from Arduino)
602     RFID_label = uilabel(sensorViewFig,'Text', 'RFID Value:', '
    Position',[390 290 200 25],'FontSize',16);
603     RFID_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390
    270 200 25],'Tag','RFID');
604
605     %Axes that will show data points
606     dataAxes = uiaxes(sensorViewFig,'Position',  [20, 20, 340,
    440],'ButtonDownFcn',@dataAxesCallback); %Create Axes
607     dataAxes.XLim = [0 8.5]; %Set axes limits
608     dataAxes.YLim = [0 11];
609     dataAxes.PickableParts = 'all'; %Make axes clickable
610     dataAxes.HitTest = 'on'; %Make axes clickable
611     dataAxes.NextPlot = 'replacechildren';
612
613     %Scroll buttons and data point number. Scrolling increases/
    decreases
614     %data point number.
615     scroll_left = uibutton(sensorViewFig,'push','Text','<<','
    Position',[390, 50, 90, 22],'ButtonPushedFcn',{
    @leftScrollCallback,dataAxes});
616     scroll_right = uibutton(sensorViewFig,'push','Text','>>','
    Position',[490, 50, 90, 22],'ButtonPushedFcn',{
    @rightScrollCallback,dataAxes});
617     dpn_val = uilabel(sensorViewFig,'Text', 'Data Point Number: ',
    'Position',[390 75 200 25],'Tag','DPN');
618
619     %Plot the sensor data on the axes
620     update_sensor_data(dataAxes);
621
622 end
623
624  function runCallback(obj,~)
625     %This is the callback function for the run button. It starts
    running
626     %whatever Gcode sequence is currently in the buffer by sending
    a start
627     %signal to the Arduino. The Arduino responds with some initial
    location
628     %and sensor data, then MATLAB knows that it can send the first
    gcode
629     %command
630
631     global gcode_buffer;
632     global s;
```

```
633     global serial_on;
634     global prgm_end;
635 
636     %Program end set to 0 means the program is starting.
637     prgm_end = 0;
638     serial_on = 1;
639 
640     %If theres stuff in the buffer and run has been turned on, send
        the
641     %start signal
642     if length(gcode_buffer) > 0 && obj.Value == 1
643         %serial_on = 0;
644         t = timer('TimerFcn',@timer_trash,'StopFcn',{@send_gcode,s
        },'StartDelay',0.05);
645         start(t);
646     %Otherwise, turn the button back off
647     else
648         obj.Value = 0;
649     end
650 
651     %If the button has been unpressed, press it again
652     if obj.Value == 0
653         obj.Value = 1;
654     end
655 
656     %If there's nothing in the buffer, the program can be ended by
657     %unpressing manually
658     if length(gcode_buffer) == 0 && obj.Value == 1
659         obj.Value = 0;
660     end
661 
662 end
663 
664 function leftScrollCallback(~,~,dataAxes)
665     %This is the callback function for the left scroll button in
        the sensor
666     %view window. This decreases the datapoint number so that you
        can look
667     %at the next data point on the list
668 
669     global sensor_data;
670     sds = size(sensor_data);
671 
672     %Find the data point colored blue
673     last = findobj(dataAxes.Children,'Color',[0 0 1]);
674     if size(last) > 0
675         %get the data point number from the tags
676         id = last.Tag;
677         newid = str2num(id)-1; %decrease data point number (go to
        next point)
678         %If the data point number is 0, go to the highest one
679         if newid == 0
680            newid = sds(1);
681         end
682         %Update the sensor view with the new selected data point
683         update_sensor_view(num2str(newid),dataAxes);
684     else
685         %If none had been selected previously, just select the
```

```
    first one
686         update_sensor_view('1',dataAxes);
687     end
688 end
689 
690 function rightScrollCallback(~,~,dataAxes)
691     %This is the callback function for the right scroll button in
    the sensor
692     %view window. This increases the datapoint number so that you
    can look
693     %at the next data point on the list
694 
695     global sensor_data;
696     sds = size(sensor_data);
697 
698     %Find the data point colored blue
699     last = findobj(dataAxes.Children,'Color',[0 0 1]);
700     if size(last) > 0
701         %Grab the data point number and increase it by one
702         id = last.Tag;
703         newid = str2num(id)+1;
704         %If the new data point number is too large, go back to the
    first
705         %one
706         if newid == sds(1)+1
707             newid = 1;
708         end
709         %Update the sensor view with the new selected data point
710         update_sensor_view(num2str(newid),dataAxes);
711     else
712         %If no point had been selected previously, just select
    number 1
713         update_sensor_view('1',dataAxes);
714     end
715 end
716 
717 function dataPointCallback(self,~,dataAxes)
718     %This is the callback function for clicking on a data point. It
     simply
719     %grabs the tag from itself and then passes it on to the
720     %update_sensor_view function so that it can select this new
    point.
721 
722     id = self.Tag;
723 
724     update_sensor_view(id,dataAxes);
725 
726 end
727 
728 function dirSendCallback(~,~,dir_ser,ser_get)
729     % Callback function for the send button in the direct serial
    comm
730     % section.
731 
732     global s;
733 
734     % For every item in the serial send bar, send it on serial and
    delete
```

```
735     % the line from the text area.
736     for idx = 1:1:length(dir_ser.Value)
737         disp(dir_ser.Value{idx});
738         ser_get.Value{end+1} = '';
739         ser_get.Value{end+1} = sprintf('Outgoing Serial');
740         ser_get.Value{end+1} = sprintf(dir_ser.Value{idx});
741         scroll(ser_get,'bottom');
742 
743         % Only send if its not an empty line.
744         if ~isempty(dir_ser.Value{idx})
745             writeline(s,dir_ser.Value{idx});
746             dir_ser.Value{idx} = '';
747         end
748     end
749 end
750 
751 function serGetCallback(self,~)
752     %Scroll the serial monitor to the bottom
753 
754     scroll(self,'bottom');
755 end
756 
757 function dataAxesCallback(~,~)
758     %This callback function does nothing
759 end
760 % ======================== %
761 
762 
763 % === UPDATE FUNCTIONS === %
764 function update_clear_btn()
765     % This function updates the display on the buffer clear button
    that
766     % tells you how many lines are in the buffer
767 
768     global gui;
769     global gcode_buffer;
770     children = get(gui,'Children');
771     % Find the button by Tag and then set the number in the text to
    include
772     % new size of gcode_buffer
773     for i = 1:1:length(children)
774         if children(i).Tag == 'C'
775             children(i).Text = ['Clear Buffer (' int2str(length(
    gcode_buffer)) ')'];
776         end
777     end
778 
779     if length(gcode_buffer) == 0
780         run_btn = findobj(gui.Children,'Tag','RUN');
781         run_btn.Value = 0;
782     end
783 
784 end
785 
786 function update_bufferfig()
787     %This function updates the contents of the text box in the
    external
788     %figure which shows the GCode buffer.
```

```
789
790     global bufferfig;
791     global gcode_buffer;
792
793     %If the gcode buffer has stuff in it and the figure exists, set
        the new
794     %value of the text box
795     if length(gcode_buffer) > 0 && sum(size(findobj('Value',
    bufferfig))) > 0
796         bufferfig.Value = flip(gcode_buffer);
797     end
798
799 end
800
801 function update_sensor_data(dataAxes)
802     %This function updates the plot in the sensor view window. It
    plots all
803     %of the newest sensor data on the axes.
804
805     global sensor_data;
806
807     %Set the axes to replace old plots with new ones, then plot a
    garbage
808     %point
809     dataAxes.NextPlot = 'replacechildren';
810     plot(dataAxes,-1,-1,'ro');
811
812     %Set axes to ADD all new plots
813     dataAxes.NextPlot = 'add';
814
815     sd = size(sensor_data);
816
817     %Plot all of the points
818     for i = 1:1:sd(1)
819         plot(dataAxes,sensor_data(i,1),sensor_data(i,2),'ro','Tag',
    num2str(i),'ButtonDownFcn',{@dataPointCallback,dataAxes});
820     end
821
822 end
823
824 function update_sensor_view(id,dataAxes)
825     %This function updates the sensor view window given a newly
    selected
826     %data point. The data point with data point number 'id' is
    turned blue
827     %and its data is shown on the right hand side of the window.
828
829     global sensorViewFig;
830     global sensor_data;
831
832     %Find the point that's colored blue
833     last = findobj(dataAxes.Children,'Color',[0 0 1]);
834
835     %If there was a blue point, turn it back to red
836     if size(last) > 0
837         last.Color = [1 0 0];
838         last.MarkerSize = 6;
839     end
```

```
840
841     %Get the point with data point number id and turn it blue
842     curr = findobj(dataAxes.Children,'Tag',id);
843     curr.Color = [0 0 1];
844     curr.MarkerSize = 10;
845
846     nid = str2num(id);
847
848     %Grab all of the data display labels
849     coord_lbl = findobj(sensorViewFig.Children,'Tag','COORD');
850     lsv_lbl = findobj(sensorViewFig.Children,'Tag','LSV');
851     rfid_lbl = findobj(sensorViewFig.Children,'Tag','RFID');
852     dpn_lbl = findobj(sensorViewFig.Children,'Tag','DPN');
853
854     %get all of the values to display from the sensor data (depends
        on id)
855     x = num2str(sensor_data(nid,1));
856     y = num2str(sensor_data(nid,2));
857     z = num2str(sensor_data(nid,3));
858     lsv = num2str(sensor_data(nid,7));
859     rfid = num2str(sensor_data(nid,8));
860
861     %Change the text of the labels to match the data gathered from
862     %sensor_data
863     coord_lbl.Text = ['X: ' x '    Y: ' y '    Z: ' z];
864     lsv_lbl.Text = lsv;
865     rfid_lbl.Text = rfid;
866     dpn_lbl.Text = ['Data Point Number: ' id];
867
868 end
869 % ======================== %
870
871
872 % === SERIAL FUNCTIONS === %
873 function readSerialData(src,~)
874     %This is the callback function for the serial device. This is
        called
875     %when there are bytes available.
876
877     global sensor_data;
878     global gui;
879     global sensorViewFig;
880     global serial_on;
881     values = zeros(1,8);
882
883     %If there are bytes available and serial is enabled, grab all
        of the
884     %new data.
885     serial_on = 1;
886     if src.NumBytesAvailable > 0 && serial_on == 1
887         values = getSerial(src);
888         sensor_data = [sensor_data; values];
889
890         %If the sensor view window is open, update the axes.
891         if size(findobj(sensorViewFig,'Type','figure')) > 0
892             dataAxes = findobj(sensorViewFig.Children,'Type','axes'
        );
893             if size(dataAxes) > 0
```

```
                update_sensor_data(dataAxes);
            end
        end
    end

end

function [values] = getSerial(src)
    %This function actually accesses the serial data and parses it
    into the
    %sensor data matrix

    global gcode_buffer;
    global gui;

    character = ' ';
    data = '';
    current = '';
    j = -1;
    lastj = -1;
    values = zeros(1,8);

    %Keep getting data until it hits a G (end of code) or a # (end
    of transmission);
    while character ~= 'G' && character ~='#'
        character = read(src,1,"char");
        data = [data  character];
    end

    %Parse data and place into sensor_data depending on the letter
    %preceding it.
    for i = 1:1:length(data)
        if data(i) == 'X' %X location
            j = 1;
        elseif data(i) == 'Y' %Y location
            j = 2;
        elseif data(i) == 'Z' %Z location
            j = 3;
        elseif data(i) == 'A' %X location
            j = 4;
        elseif data(i) == 'B' %Y location
            j = 5;
        elseif data(i) == 'C' %Z location
            j = 6;
        elseif data(i) == 'L' %Light sensor value
            j = 7;
        elseif data(i) == 'R' %RFID value
            j = 8;
        elseif data(i) == 'G' %Send another Gcode command
            j = -1;
            %For some reason, the serial write command cannot be
    used
            %within the serial read callback, so short timers have
    to be
            %used.
            if ~isempty(gcode_buffer)
                t = timer('TimerFcn',@timer_trash,'StopFcn',{
    @send_gcode,src},'StartDelay',0.1);
```

```
947                 start(t);
948             end
949         elseif data(i) == '#'
950             while character ~= 'G'
951                 character = read(src,1,"char");
952             end
953             character = read(src,1,"char");
954             j=-1;
955         end
956
957         %If theres a new letter, put all of the previous numbers
    into the
958         %sensor values
959         if j ~= lastj && lastj ~= -1
960             values(1,lastj) = str2double(current(2:end));
961             current = '';
962         end
963
964         %If j is unchanged (same letter), keep adding the
    information
965         if j ~= -1
966             current = [current data(i)];
967         end
968
969         lastj = j;
970     end
971
972     ser_get = findobj(gui,'Tag','SER');
973
974     if sum(values(1:3)) ~= 0
975         disp("Incoming Location Data:");
976         disp(values(1:3));
977         ser_get.Value{end+1} = sprintf('Incoming Location Data:');
978         ser_get.Value{end+1} = sprintf(['X: ' num2str(values(1)) '
    Y: ' num2str(values(2)) '  Z: ' num2str(values(3))]);
979         scroll(ser_get,'bottom');
980     end
981
982     if sum(values(4:6)) ~= 0
983         disp("Incoming Thread Length Data:");
984         disp(values(4:6));
985         ser_get.Value{end+1} = sprintf('Incoming Thread Length Data
    :');
986         ser_get.Value{end+1} = sprintf(['A: ' num2str(values(4)) '
    B: '  num2str(values(5)) '  C: ' num2str(values(6))]);
987         scroll(ser_get,'bottom');
988     end
989
990 end
991
992 function send_gcode(~,~,src)
993     %This function sends 1 line of gcode through serial to the
    Arduino.
994     %Because of the nature of the serial read callback, this
    function has
995     %to be set up as a callback function for a timer.
996
997     global gcode_buffer;
```

```
998     global serial_on;
999     global gui;
1000    global prgm_end;
1001
1002    % Grab the serial monitor
1003    ser_get = findobj(gui,'Tag','SER');
1004
1005    serial_on = 0;
1006
1007    %If the gcode buffer has stuff in it and theres nothing more to
        read,
1008    %send the stuff.
1009    if length(gcode_buffer) > 0 && src.NumBytesAvailable == 0
1010         %get rid of all of the empty lines or line breaks
1011         while (sum(gcode_buffer{end} == ';') == 1 && length(
       gcode_buffer{end}) == 1) || length(gcode_buffer{end}) == 0
1012              gcode_buffer(end) = [];
1013         end
1014         writeline(src,gcode_buffer{end});
1015         disp("Outgoing GCode:");
1016         disp(gcode_buffer{end});
1017         % Print gcode to serial monitor
1018         ser_get.Value{end+1} = '';
1019         ser_get.Value{end+1} = sprintf('Outgoing Gcode');
1020         ser_get.Value{end+1} = sprintf(gcode_buffer{end});
1021         scroll(ser_get,'bottom');
1022         gcode_buffer(end) = [];
1023         update_clear_btn();
1024    end
1025
1026    %Only turn the serial back on if the program is not ended
1027    if prgm_end == 0
1028         serial_on = 1;
1029    end
1030
1031    update_bufferfig();
1032
1033 end
1034
1035
1036 % === PROGRAM FLOW FUNCTIONS === %
1037 function timer_trash(~,~)
1038    %Timers have to have a function to be executed when they start.
        Nothing
1039    %needed to happen at the start of the timers in this program,
       so this
1040    %function is empty.
1041 end
1042
1043 function end_program(~,~)
1044    %This function is run when the current gcode sequence has run
       its
1045    %course. It deactivates serial, deactivates the run button, and
        sets
1046    %the program end flag to true.
1047
1048    global gui;
1049    global serial_on;
```

```
1050     global prgm_end;
1051 
1052     run_btn = findobj(gui.Children,'Tag','RUN');
1053     run_btn.Value = 0;
1054     serial_on = 0;
1055     prgm_end = 1;
1056 
1057 end
```

8.3 Calculating Thread Lengths in Arduino

Current and target thread lengths are calculated using Pythagorean's theorem within the Arduino. First we must find the length of the projection of the thread connected to the payload. With that length, it is possible to calculate the necessary thread length by using the height of the triangle that is created between the thread, projection and the pylon.

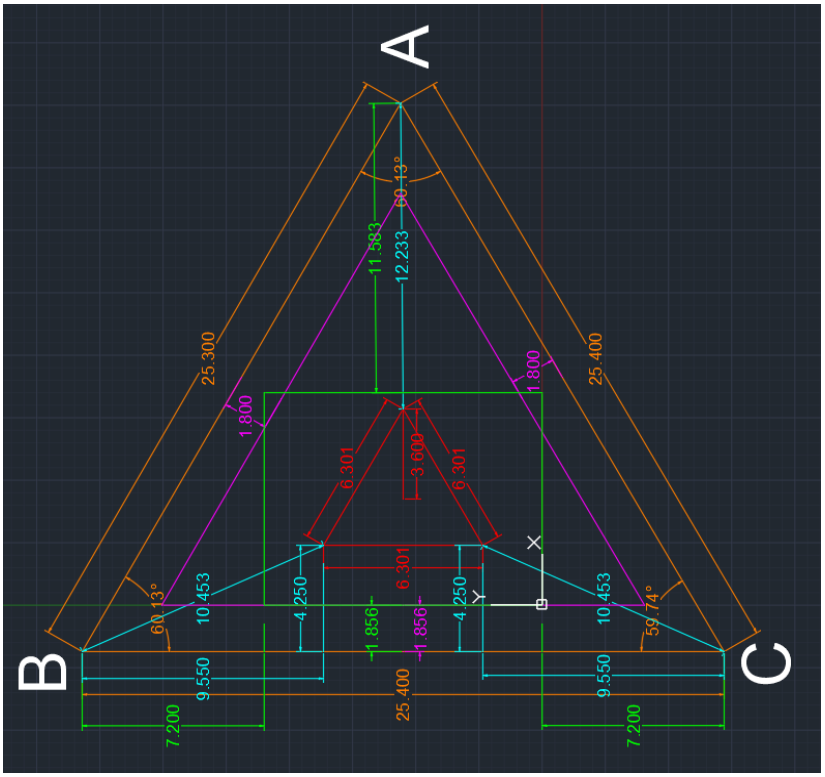

Figure 48: Calculating projection of thread length based on known values.

After the projections in light blue in Figure 48 are calculated, the actual thread length can be calculated by using the Pythagorean's theorem with the difference in height from the top of the pylon to the payload and the projection length. This is very evident when you look at Figure 49.

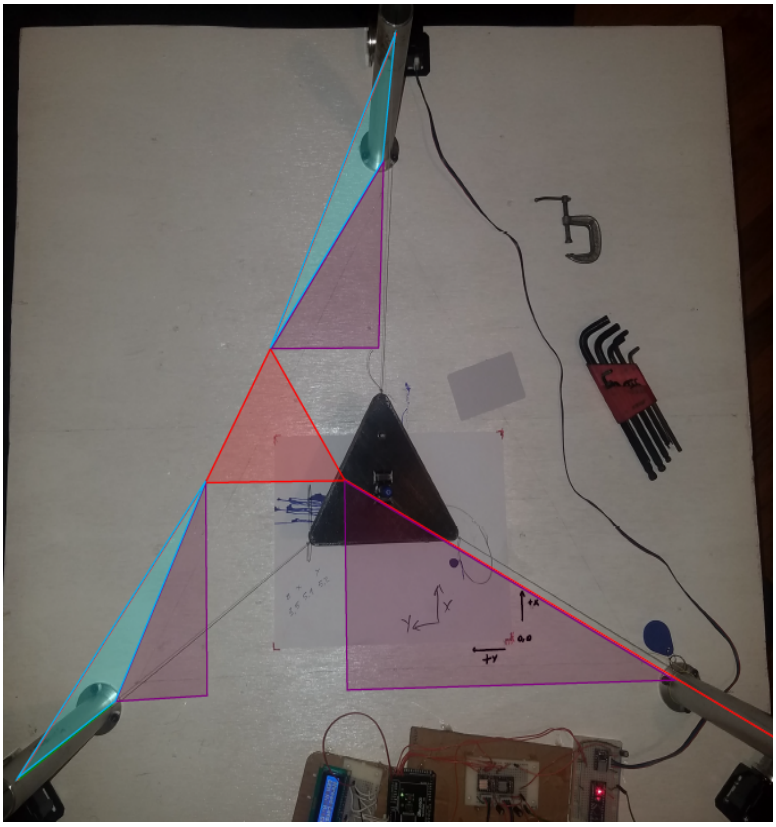

Figure 49: Calculating thread length based on projection of thread and height of the blue triangle.