Developer Guide

Authors:
Benjamin Adams
Miles Drake
Trevor Horine
Felipe Orrico Scognamiglio

March 5, 2021
Instructor: Matthew Shuman
## Contents

1 **System Overview**  
   1.1 System Block Diagram ........................................... 5

2 **Electrical Specifications**  
   2.1 System Interface Table ........................................... 6  
   2.2 Schematics .......................................................... 7

3 **User Guide**  
   3.1 Setup ............................................................... 9  
      3.1.1 Hardware ..................................................... 9  
      3.1.2 Software - Arduino ......................................... 10  
      3.1.3 Software - MATLAB ......................................... 10  
   3.2 Operation .......................................................... 11  
      3.2.1 MATLAB ..................................................... 11  
      3.2.2 Arduino ..................................................... 11

4 **Design Artifacts**  
   4.1 Arduino Code ..................................................... 12  
   4.2 MATLAB Code ...................................................... 13  
   4.3 Mechanical Subsystem ............................................ 15  
   4.4 Payload ............................................................ 17  
      4.4.1 Triangular Payload .......................................... 17  
      4.4.2 Rectangular Payload ......................................... 18  
   4.5 PCB Enclosure ..................................................... 20

5 **PCB Information**  
   5.1 PCB Block Diagram ............................................... 27  
   5.2 PCB Schematics .................................................... 28  
   5.3 PCB Interface Table ............................................... 29  
   5.4 Eagle Layout and Board Dimensions ............................. 30  
   5.5 PCB Mechanical Drawing .......................................... 31  
   5.6 Board Profile ..................................................... 32  
   5.7 Top Silkscreen ..................................................... 33  
   5.8 Top Copper ........................................................ 34  
   5.9 Top Soldermask .................................................... 35  
   5.10 Top Soldermask and Silkscreen ................................ 36  
   5.11 Bottom Silkscreen ............................................... 37  
   5.12 Bottom Copper .................................................. 38  
   5.13 Bottom Soldermask ............................................... 39  
   5.14 Bottom Soldermask and Silkscreen ............................ 40  
   5.15 Top Gerbers ...................................................... 41  
   5.16 Bottom Gerbers .................................................. 42  
   5.17 3D Model .......................................................... 43  
   5.18 Assembled Board ................................................ 44

6 **Parts Information and Bill of Material**  

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

![Image of Interface Table]

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

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.

Figure 6: CAD image of relative hardware measurements with a triangular payload
The final hardware should look like Figure 7.

![Figure 7: Final Hardware- top view](image)

### 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). Its 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.

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

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:

\[X<\text{x-location}>Y<\text{y-location}>Z<\text{z-location}>A<\text{thread-len-a}>B<\text{thread-len-b}>C<\text{thread-len-c}>L<\text{light-level}>R<\text{rfid-data}>G\]

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.

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 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 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 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
4.4.2 Rectangular Payload

Figure 16: Triangular payload assembly- exploded view

Figure 17: Rectangular payload- bottom part
Figure 18: Rectangular payload- top part
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 halves 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

<table>
<thead>
<tr>
<th>Interface</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USB</strong></td>
<td>Supplies 5V power to Arduino. Also used a serial comunication between computer and Arduino</td>
</tr>
<tr>
<td><strong>12V Power</strong></td>
<td>12V, 3.5 A power supplied to board via a barrel jack for the motor drivers.</td>
</tr>
<tr>
<td><strong>Sensor Inputs</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Motors</strong></td>
<td>Three 4 pin conectors that conect the motors to the PCB. Each of these conectors is attached to a different motor controller</td>
</tr>
<tr>
<td><strong>Fan Power</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Sensor Power</strong></td>
<td>The remaing two pins in the 5 pin sensor conector to carry 5V and ground to the sensors on the payload.</td>
</tr>
<tr>
<td><strong>LCD</strong></td>
<td>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.)</td>
</tr>
</tbody>
</table>

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 from 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.
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.
Figure 42: This is a rendering of the 3D model produced from 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.
### Parts Information and Bill of Material

<table>
<thead>
<tr>
<th>Part</th>
<th>Value and units</th>
<th>Quantity</th>
<th>Price</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PCBs</strong></td>
<td></td>
<td></td>
<td></td>
<td>让大家切为人及地在机内有撞击的卫. <strong>Datashet was found for a board that appears to be same as there was not a datashet on the amazon page.</strong> <strong>DRV8825 Chip data sheet</strong></td>
</tr>
<tr>
<td>Stepper drivers</td>
<td></td>
<td>5</td>
<td>9.99</td>
<td>Will use stepper drive from version classes. 64 female, 68 male</td>
</tr>
<tr>
<td>Pin Headers</td>
<td></td>
<td>150</td>
<td>0</td>
<td>Used for debug and display</td>
</tr>
<tr>
<td>Barrel Jack</td>
<td></td>
<td>1</td>
<td>0.53</td>
<td>Used to connect wall wart to board</td>
</tr>
<tr>
<td>LCD</td>
<td></td>
<td>1</td>
<td>3.25</td>
<td><strong>Arduin Mega 2560</strong> <strong>Arduin</strong>, this is the brain</td>
</tr>
<tr>
<td>SSB-XH(LF)(SN) Conector</td>
<td></td>
<td>1</td>
<td>0.31</td>
<td>Conector to attach light sensor and RFID</td>
</tr>
<tr>
<td>S4B-XH-A(LF)(SN) Conector</td>
<td></td>
<td>3</td>
<td>0.81</td>
<td>Conectors to attach motors to board each one is a 4 pin right angle conector</td>
</tr>
<tr>
<td>Arduino Mega 2560 Arduino</td>
<td></td>
<td>1</td>
<td>16.99</td>
<td><strong>Tin on the 12 volt power source before the motor drivers to help with current draw</strong></td>
</tr>
<tr>
<td>POTENTIOMETER 10k</td>
<td></td>
<td>1</td>
<td>2.9</td>
<td><strong>Used for contrast on LCD</strong></td>
</tr>
<tr>
<td>Resistor</td>
<td></td>
<td>220</td>
<td>0</td>
<td><strong>Resistor for LCD</strong></td>
</tr>
<tr>
<td>Resistor 10K</td>
<td></td>
<td>1</td>
<td>0</td>
<td><strong>Resistor for voltage divider with photocell to measure light level</strong></td>
</tr>
<tr>
<td>Capacitor 100μ</td>
<td></td>
<td>3</td>
<td>0.69</td>
<td><strong>Used on the 12 volt power source before the motor drivers to help with current draw</strong></td>
</tr>
<tr>
<td>Capacitor 10μ</td>
<td></td>
<td>3</td>
<td>0.66</td>
<td><strong>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.</strong></td>
</tr>
<tr>
<td>Fan</td>
<td></td>
<td>2</td>
<td>6</td>
<td><strong>40mmx40mmx10mm fan should it be needed to keep motor drivers from heating up to much.</strong></td>
</tr>
<tr>
<td>Boards</td>
<td></td>
<td>5</td>
<td>10</td>
<td><strong>Minimum order quantity is 5 baords.</strong></td>
</tr>
<tr>
<td>PCB Enclosure</td>
<td></td>
<td>1</td>
<td>5</td>
<td><strong>3D printed enclosure for the PCB</strong></td>
</tr>
<tr>
<td>M3, 35mm screws</td>
<td></td>
<td>4</td>
<td>2.48</td>
<td><strong>Screws used to hold PCB in place and the enclosure closed.</strong></td>
</tr>
<tr>
<td><strong>PAYLOAD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photocell</td>
<td></td>
<td>1</td>
<td>0.89</td>
<td><strong>Photocell to be used in voltage divider for measuring light level</strong></td>
</tr>
<tr>
<td>RFID breakout</td>
<td></td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Female Cable Connector Housing</td>
<td></td>
<td>1</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>Ribbon Cable</td>
<td></td>
<td>3 ft, 6 wire 24awg</td>
<td>1</td>
<td>5.05</td>
</tr>
<tr>
<td>#2 screw- RFID board mount</td>
<td></td>
<td>2</td>
<td>0.14</td>
<td><strong>Used to mount board to payload</strong></td>
</tr>
<tr>
<td>#6-32 1/2&quot; set screw</td>
<td></td>
<td>1</td>
<td>0.19</td>
<td><strong>Used to secure writing utensil</strong></td>
</tr>
<tr>
<td>Payload Enclosure- Top</td>
<td></td>
<td>1</td>
<td>5.44</td>
<td><strong>3D printed enclosure for the payload sensors and writing implemt through Corvallis3d.</strong></td>
</tr>
<tr>
<td>Payload Enclosure- Bottom</td>
<td></td>
<td>1</td>
<td>14.98</td>
<td><strong>3D printed enclosure for the payload sensors and writing implemt through Corvallis3d.</strong></td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stepper motors</td>
<td></td>
<td>3</td>
<td>24.99</td>
<td><strong>Datashet info on the amazon page</strong></td>
</tr>
<tr>
<td>Stepper Motor Brackets</td>
<td></td>
<td>5</td>
<td>13.99</td>
<td></td>
</tr>
<tr>
<td>2&quot; Aluminum Round Stock</td>
<td></td>
<td>1 ft</td>
<td>0</td>
<td><strong>Used scrap aluminum</strong></td>
</tr>
<tr>
<td>Aluminum Tube Stock 1/8&quot; wall, 1&quot; nominal (ID)</td>
<td>3.375 ft</td>
<td>0</td>
<td><strong>Used scrap aluminum</strong></td>
<td></td>
</tr>
<tr>
<td>15/32&quot; Plywood sheet</td>
<td></td>
<td>3.5' x 3.5'</td>
<td>0</td>
<td><strong>Used scrap wood</strong></td>
</tr>
<tr>
<td>1/4&quot;-20 x 1.25&quot; Socket Head Cap Screw</td>
<td>9</td>
<td>1.71</td>
<td><strong>Pylon attachment hardware</strong></td>
<td></td>
</tr>
<tr>
<td>1/4&quot; Washer</td>
<td></td>
<td>9</td>
<td>0.63</td>
<td><strong>Pylon attachment hardware</strong></td>
</tr>
<tr>
<td>1/4&quot;-20 Nut</td>
<td></td>
<td>9</td>
<td>0.81</td>
<td><strong>Pylon attachment hardware</strong></td>
</tr>
<tr>
<td>Fishing line</td>
<td></td>
<td>0</td>
<td>0</td>
<td><strong>Monofilament 9lb test fishing line was used as a strong light weight string to connect the payload to the motors.</strong></td>
</tr>
</tbody>
</table>

**Total Cost**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Total Cost</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>138.55</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Percentage per Person</th>
<th>Total Manhours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trevor</td>
<td>33.42</td>
</tr>
<tr>
<td>Ben</td>
<td>25.99</td>
</tr>
<tr>
<td>Miles</td>
<td>19.06</td>
</tr>
<tr>
<td>Felipe</td>
<td>21.53</td>
</tr>
</tbody>
</table>

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

```c
/*
 * 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 /////////////
const double A_PAPER_DISTANCE = 11.583;
```
const double B_PAPER_DISTANCE = 7.2;
const double C_PAPER_DISTANCE = 7.2;
const double HEIGHT_PYON_A = 11.8;
const double HEIGHT_PYON_B = 11.8;
const double HEIGHT_PYON_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 P_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;
bool M6_SAVE_Speed = 0;
bool M6_EN = false;

// //////////////////////// VARIABLES //////////////////////////
// ////////////////////////// FLAGS /////////////////////////////
bool UNIT_MODE = true; // true = inches, false = mm;
bool ABSOLUTE_COORDINATES = true; // true = absolute, false = incremental
bool ON_MOVE = false;
bool M2_EN = false;

#include "AccelStepper.h"
#include "MultiStepper.h"

// change pins here to reflect correct pins
#define dirPinA 25
#define stepPinA 23
#define dirPinB 29
#define stepPinB 27
#define dirPinC 33
#define stepPinC 31
#define motorInterfaceType 1

AccelStepper stepperA; // = AccelStepper(motorInterfaceType, stepPinA, dirPinA);
AccelStepper stepperB; // = AccelStepper(motorInterfaceType, stepPinB, dirPinB);
AccelStepper stepperC; // = AccelStepper(motorInterfaceType, stepPinC, dirPinC);

MultiStepper steppers = MultiStepper();

void G0_cmd(){
    if (ON_MOVE || M6_EN){
        Serial.println("ON_MOVE");
        return;
    }
    double x_coord = CURRENT_X;
    double y_coord = CURRENT_Y;
}
double z_coord = CURRENT_PAYLOAD_HEIGHT;

if(Commands.availableValue('X')){
    x_coord = Commands.GetValue('X');
}

if(Commands.availableValue('Y')){
    y_coord = Commands.GetValue('Y');
}

if(Commands.availableValue('Z')){
    z_coord = Commands.GetValue('Z');
}

int f_speed = 100; //maximum speed

if (!UNIT_MODE){ //translate mm to in
    x_coord = mm_to_in(x_coord);
    y_coord = mm_to_in(y_coord);
    z_coord = mm_to_in(z_coord);
}

if (!ABSOLUTE_COORDINATES){ //incremental mode
    x_coord += CURRENT_X;
    y_coord += CURRENT_Y;
    z_coord += CURRENT_PAYLOAD_HEIGHT;
}

//checking for boundaries
if (x_coord > 8.5)
    x_coord = 8.5;
if (y_coord > 11)
    y_coord = 11;
if (z_coord > 11)
    z_coord = 11;

//update global values
Y_ADDRESS_TARGET = y_coord;
X_ADDRESS_TARGET = x_coord;
Z_ADDRESS_TARGET = z_coord;
F_PERCENT_SPEED = f_speed;

update_lcd(0);
go(x_coord,y_coord,z_coord, f_speed);

void G1_cmd(){

    if (ON_MOVE || M6_EN) return;

    double x_coord = CURRENT_X;
    double y_coord = CURRENT_Y;
    double z_coord = CURRENT_PAYLOAD_HEIGHT;
    double f_speed_in = 4.5;
    int f_speed = 100; //maximum speed

    if(Commands.availableValue('X')){
        x_coord = Commands.GetValue('X');
    }
if(Command. availableValue('Y')){
    y_coord = Command. getValue('Y');
}
if(Command. availableValue('Z')){
    z_coord = Command. getValue('Z');
}
if(Command. availableValue('F')){
    f_speed_in = Command. getValue('F');
}
if (!UNIT. MODE) { // translate mm to in
    x_coord = mm_to_in(x_coord);
    y_coord = mm_to_in(y_coord);
    z_coord = mm_to_in(z_coord);
    f_speed_in = mm_to_in(f_speed_in);
}
if (!ABSOLUTE. COORDINATES) { // incremental mode
    x_coord += CURRENT_X;
    y_coord += CURRENT_Y;
    z_coord += CURRENT_PAYLOAD_HEIGHT;
}
// checking for boundaries
if (x_coord > 8.5)
    x_coord = 8.5;
if (y_coord > 11)
    y_coord = 11;
if (z_coord > 11)
    z_coord = 11;

f_speed_in = (f_speed_in/4.5)*100;
if (f_speed_in > 100)
    f_speed = 100;
else if (f_speed_in <= 1)
    f_speed = 1;
else
    f_speed = int(f_speed_in);

// update global values
Y_ADDRESS. TARGET = y_coord;
X_ADDRESS. TARGET = x_coord;
Z_ADDRESS. TARGET = z_coord;
F_PERCENT. SPEED = f_speed;
update_lcd(1);
go(x_coord, y_coord, z_coord, f_speed);

void G20. cmd(){
    if (ON_MOVE || M6. EN) return;
    UNIT. MODE = true;
    update_lcd(20);
}
void G21_cmd()
{
    if (ON_MOVE || M6_EN) return;
    UNIT_MODE = false;
    update_lcd(21);
}

void G90_cmd()
{
    if (ON_MOVE || M6_EN) return;
    ABSOLUTE_COORDINATES = true;
    update_lcd(90);
}

void G91_cmd()
{
    if (ON_MOVE || M6_EN) return;
    ABSOLUTE_COORDINATES = false;
    update_lcd(91);
}

void M2_cmd()
{
    update_lcd(2);
    if (ON_MOVE){
        stepperA.stop();
        stepperB.stop();
        stepperC.stop();
        ON_MOVE = false;
        ABSOLUTE_COORDINATES = true;
        M2_EN = true;
        M6_EN = false;
    }
}

void M6_cmd()
{
    update_lcd(6);
    if (!M6_EN){ //m6 first time
        //stop
        if (ON_MOVE){
            stepperA.stop();
            stepperB.stop();
            stepperC.stop();
            ON_MOVE = false;
            ABSOLUTE_COORDINATES = true;
            M6_EN = true;
        }
        double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
        double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
        double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;
    }
CURRENT_THREAD_LEN_A += actual_a;
CURRENT_THREAD_LEN_B += actual_b;
CURRENT_THREAD_LEN_C += actual_c;

M6_SAVE_LEN_A = CURRENT_THREAD_LEN_A;
M6_SAVE_LEN_B = CURRENT_THREAD_LEN_B;
M6_SAVE_LEN_C = CURRENT_THREAD_LEN_C;
M6_SAVE_X = X_ADDRESS_TARGET;
M6_SAVE_Y = Y_ADDRESS_TARGET;
M6_SAVE_Z = Z_ADDRESS_TARGET;

if (M6_SAVE_X == -1 || M6_SAVE_Y == -1 || M6_SAVE_Z == -1){
    M6_SAVE_X = CURRENT_X;
    M6_SAVE_Y = CURRENT_Y;
    M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
}
if (M6_SAVE_X == 0 && M6_SAVE_Y == 0 && M6_SAVE_Z == 0){
    M6_SAVE_X = CURRENT_X;
    M6_SAVE_Y = CURRENT_Y;
    M6_SAVE_Z = CURRENT_PAYLOAD_HEIGHT;
}
M6_SAVE_Speed = F_PERCENT_SPEED;

go_block(HOME_X, HOME_Y, HOME_Z, 100);
}
else {
    go_len_block(M6_SAVE_LEN_A, M6_SAVE_LEN_B, M6_SAVE_LEN_C, 100);
    M6_EN = false;
    // go(M6_SAVE_X, M6_SAVE_Y, M6_SAVE_Z, M6_SAVE_Speed);
    /*
     * M6_SAVE_LEN_A = 0;
     * M6_SAVE_LEN_B = 0;
     * M6_SAVE_LEN_C = 0;
     * M6_SAVE_X = -1;
     * M6_SAVE_Y = -1;
     * M6_SAVE_Z = -1;
     * M6_SAVE_Speed = 0;*
    */
}
}
void C1_cmd(){ //Write value of X,Y,Z to the Serial Connection
    //if (ON_MOVE || M6_EN) return;
    Serial.write("X");
    Serial.print(CURRENT_X);
    Serial.write("Y");
    Serial.print(CURRENT_Y);
    Serial.write("Z");
    Serial.print(CURRENT_PAYLOAD_HEIGHT);
    Serial.write("#");
    update_lcd(57);
    delay(500);
void C2_cmd() { //Write thread lengths to Serial
  //if (ON_MOVE || M6_EN) return;

  Serial.write("A");
  Serial.print(CURRENT_THREAD_LEN_A);
  Serial.write("B");
  Serial.print(CURRENT_THREAD_LEN_B);
  Serial.write("C");
  Serial.print(CURRENT_THREAD_LEN_C);
  Serial.write("#");
  update_lcd(58);
  delay(500);
}

// ///////////////////////////////////////////////////G-Code// ///////////////////////////////////////////////////

// ///////////////////////////////////////////////////////////////////////////////////////////////////THREAD CALCULATIONS// ///////////////////////////////////////////////////////////////////////////////////

double calculate_thread_height_var(double z){
  return HEIGHT_PYLON_A - z;
}

double calculate_thread_length_A_h(double x, double y, double z) {
  double height_created_triangle = A_PAPER_DISTANCE + 8.5 - x - PAYLOAD_CENTER_THREAD_DISTANCE_A_x;
  double side_created_triangle;
  if ((y - 5.5) < 0)
    side_created_triangle = 5.5 - y;
  else
    side_created_triangle = y - 5.5;

  double trace_length = sqrt(pow(height_created_triangle, 2) + pow(side_created_triangle, 2));
  double thread_length = sqrt(pow(trace_length,2) + pow(calculate_thread_height_var(z),2));

  return thread_length;
}

double calculate_thread_length_B_h(double x, double y, double z) {
  double height_created_triangle = PAPER_HEIGHT + B_PAPER_DISTANCE - y - PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
  double side_created_triangle = PAPER_DISTANCE_LOW_EDGE + x - PAYLOAD_CENTER_THREAD_DISTANCE_B_x;

  double trace_length = sqrt(pow(height_created_triangle,2) + pow(side_created_triangle,2));
  //double trace_length = PAPER_DISTANCE_1 - y - PAYLOAD_CENTER_THREAD_DISTANCE_B_y;

  double thread_length = sqrt(pow(trace_length,2) + pow(calculate_thread_height_var(z),2));

  return thread_length;
}
double calculate_thread_length_C_h(double x, double y, double z) {
    double height_created_triangle = C_PAPER_DISTANCE + y -
        PAYLOAD_CENTER_THREAD_DISTANCE_B_y;
    double side_created_triangle = PAPER_DISTANCE_LOW_EDGE + x -
        PAYLOAD_CENTER_THREAD_DISTANCE_C_x;
    double trace_length = sqrt(pow(height_created_triangle, 2) + pow(
        side_created_triangle, 2));
    double thread_length = sqrt(pow(trace_length, 2) + pow(
        calculate_thread_height_var(z), 2));
    return thread_length;
}

void update_lcd(int opcode) {
    lcd.clear();
    if (opcode == 0 || opcode == 1) { // G0 or G1 (displays as integer
        // for size)
        lcd.setCursor(0, 0); //1st line 1st char
        lcd.print("OPCODE: G");
        lcd.print(opcode);
        lcd.print(" F:");
        lcd.print(F_PERCENT_SPEED);
        lcd.setCursor(0, 1); //2nd Line 1st char
        double x = X_ADDRESS_TARGET;
        double y = Y_ADDRESS_TARGET;
        if (!UNIT_MODE) { // translate mm to in
            x = in_to_mm(X_ADDRESS_TARGET);
            y = in_to_mm(Y_ADDRESS_TARGET);
        }
        lcd.print("X: ");
        lcd.print(x);
        lcd.setCursor(8, 1); //2nd Line 9th char
        lcd.print("Y: ");
        lcd.print(y);
    } else if (opcode == 20) {
        lcd.setCursor(0, 0); //1st line 1st char
        lcd.print("OPCODE: G");
        lcd.print(opcode);
        lcd.setCursor(0, 1); //2nd Line 1st char
        lcd.print("Input Mode > IN");
    } else if (opcode == 21) {
        lcd.setCursor(0, 0); //1st line 1st char
        lcd.print("UNIT CONVERSION")
    }
}
lcd.print("OPCODE: G");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
lcd.print("Input Mode > MM");
}

else if (opcode == 90) {
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("OPCODE: G");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("Coord Mode > ABS");
}

else if (opcode == 91) {
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("OPCODE: G");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
            lcd.print("Coord Mode > INC");
}

else if (opcode == 2) {
    lcd.setCursor(0, 0); //1st line 1st char
    lcd.print("OPCODE: M");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
    lcd.print("END PROGRAM");
}

else if (opcode == 6) {
    lcd.setCursor(0, 0); //1st line 1st char
            lcd.print("OPCODE: G");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
            lcd.print("TOOL CHANGE");
}

else if (opcode == 57) {
    lcd.setCursor(0, 0); //1st line 1st char
            lcd.print("Sending Location");
lcd.setCursor(0, 1); //2nd Line 1st char
            lcd.print("<<<<<<<<<<<>>>>>>>>");
}

else if (opcode == 58) {
    lcd.setCursor(0, 0); //1st line 1st char
            lcd.print("Sending Threads");
lcd.setCursor(0, 1); //2nd Line 1st char
            lcd.print("<<<<<<<<<<<>>>>>>>>");
}

else {
    lcd.setCursor(0, 0); //1st line 1st char
            lcd.print("ERROR: ");
lcd.print(opcode);
lcd.setCursor(0, 1); //2nd Line 1st char
            lcd.print("OPCODE NOT FOUND");
}

// Calculates the number of steps to move from current_ to new_ length of thread.
// If the output is negative, then the motor must reduce the size of the thread instead of increasing it
int calculate_number_steps(double current_, double new_){
    int num_steps = int((new_ - current_)/INCHES_PER_STEP);
    return num_steps;
}

int go_block(double x, double y, double z, int f_speed) {
    X_ADDRESS_TARGET = x;
    Y_ADDRESS_TARGET = y;
    Z_ADDRESS_TARGET = z;
    F_PERCENT_SPEED = f_speed;

    double new_len_a = calculate_thread_length_A_h(x,y,z);
    double new_len_b = calculate_thread_length_B_h(x,y,z);
    double new_len_c = calculate_thread_length_C_h(x,y,z);

    int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, new_len_a);
    int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, new_len_b);
    int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, new_len_c);

    stepperA.setCurrentPosition(0);
    stepperB.setCurrentPosition(0);
    stepperC.setCurrentPosition(0);
    int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
    stepperA.setMaxSpeed(max_speed);
    stepperB.setMaxSpeed(max_speed);
    stepperC.setMaxSpeed(max_speed);

    long steps[] = {steps_A, -steps_B, steps_C};

    lcd.clear();
    lcd.print("Moving to >>>>");
    lcd.setCursor(0, 1);
    lcd.print(x);
    lcd.print(" ");
    lcd.print(y);
    lcd.print(" ");
    lcd.print(z);

    steppers.moveTo(steps);
    //move_to(steps_A, steps_B, steps_C);

    while(steppers.run()){
    }

    //find actual displacement
double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

//after done, update current values
CURRENT_X = x;
CURRENT_Y = y;
CURRENT_PAYLOAD_HEIGHT = z;
CURRENT_THREAD_LEN_A += actual_a;
CURRENT_THREAD_LEN_B += actual_b;
CURRENT_THREAD_LEN_C += actual_c;

return 0;
}
int go_len_block(double a, double b, double c, int f_speed) {
    //calculate number of steps and direction of movement to move
    //from current pos to new pos.
    int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, a);
    int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, b);
    int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, c);

    //Setting important info in stepper classes
    stepperA.setCurrentPosition(0);
    stepperB.setCurrentPosition(0);
    stepperC.setCurrentPosition(0);
    int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
    stepperA.setMaxSpeed(max_speed);
    stepperB.setMaxSpeed(max_speed);
    stepperC.setMaxSpeed(max_speed);

    //move the payload to location
    long steps[] = {steps_A, -steps_B, steps_C};
    steppers.moveTo(steps);

    //blocks until steppers finish moving
    while(steppers.run()){
    }

    //find actual displacement
    double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
    double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
    double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

    //after done, update current values
    CURRENT_THREAD_LEN_A += actual_a;
    CURRENT_THREAD_LEN_B += actual_b;
    CURRENT_THREAD_LEN_C += actual_c;

    X_ADDRESS_TARGET = 0;
    Y_ADDRESS_TARGET = 0;
    Z_ADDRESS_TARGET = 0;

    return 0;
```c
int go(double x, double y, double z, int f_speed) {
    X_ADDRESS_TARGET = x;
    Y_ADDRESS_TARGET = y;
    Z_ADDRESS_TARGET = z;
    F_PERCENT_SPEED = f_speed;

    // calculate new necessary thread lengths
    double new_len_a = calculate_thread_length_A_h(x, y, z);
    double new_len_b = calculate_thread_length_B_h(x, y, z);
    double new_len_c = calculate_thread_length_C_h(x, y, z);

    /* Serial.println("\n");
    Serial.print("New Length A: ");
    Serial.println(new_len_a);
    Serial.print("New Length B: ");
    Serial.println(new_len_b);
    Serial.print("New Length C: ");
    Serial.println(new_len_c);
    Serial.println("\n");
    Serial.print("CurrentLength A: ");
    Serial.println(CURRENT_THREAD_LEN_A);
    Serial.print("CurrentLength B: ");
    Serial.println(CURRENT_THREAD_LEN_B);
    Serial.print("CurrentLength C: ");
    Serial.println(CURRENT_THREAD_LEN_C);*/

    // calculate number of steps and direction of movement to move
    // from current pos to new pos.
    int steps_A = calculate_number_steps(CURRENT_THREAD_LEN_A, new_len_a);
    int steps_B = calculate_number_steps(CURRENT_THREAD_LEN_B, new_len_b);
    int steps_C = calculate_number_steps(CURRENT_THREAD_LEN_C, new_len_c);

    // Setting important info in stepper classes
    stepperA.setCurrentPosition(0);
    stepperB.setCurrentPosition(0);
    stepperC.setCurrentPosition(0);
    int max_speed = int((MAX_STEPS_PER_SEC/100)*f_speed);
    stepperA.setMaxSpeed(max_speed);
    stepperB.setMaxSpeed(max_speed);
    stepperC.setMaxSpeed(max_speed);
    // stepperA.setAcceleration(max_speed/5);
    // stepperB.setAcceleration(max_speed/5);
    // stepperC.setAcceleration(max_speed/5);
    // move the payload to location
    long steps[] = {steps_A, -steps_B, steps_C};

    lcd.clear();
    lcd.print("Moving to: F");
    lcd.setCursor(0, 1);
    lcd.print(x);
    */
```
lcd.print(" ");
lcd.print(y);
lcd.print(" ");
lcd.print(z);

steppers.moveTo(steps);

//move_to(steps_A, steps_B, steps_C);

ON_MOVE = true;

//blocks until steppers finish moving
while (steppers.run() && ON_MOVE){
    Commands.available();
}

//find actual displacement
double actual_a = stepperA.currentPosition() * INCHES_PER_STEP;
double actual_b = -(stepperB.currentPosition() * INCHES_PER_STEP);
double actual_c = stepperC.currentPosition() * INCHES_PER_STEP;

/* Serial.println(" ");
Serial.print(" DELTA A : ");
Serial.println(actual_a);
Serial.print(" DELTA B : ");
Serial.println(actual_b);
Serial.print(" DELTA C : ");
Serial.println(actual_c);

Serial.println(" \nFinal Stepper Pos: ");
Serial.print(" Stepper A: ");
Serial.println(stepperA.currentPosition());
Serial.print(" Stepper B: ");
Serial.println(-stepperB.currentPosition());
Serial.print(" Stepper C: ");
Serial.println(stepperC.currentPosition());*/

ON_MOVE = false;

//after done, update current values
if (!M2_EN && !M6_EN){
    CURRENT_X = x;
    CURRENT_Y = y;
    CURRENT_PAYLOAD_HEIGHT = z;
}
if (!M6_EN){
    CURRENT_THREAD_LEN_A += actual_a;
    CURRENT_THREAD_LEN_B += actual_b;
    CURRENT_THREAD_LEN_C += actual_c;
}
M2_EN = false;

X_ADDRESS_TARGET = 0;
Y_ADDRESS_TARGET = 0;
Z_ADDRESS_TARGET = 0;

/* 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  }
sensorVal = analogRead(sensor);
lightLevel = (sensorVal/1023)*10;
Serial.write("L");
Serial.print(lightLevel);

boolean success = false;
uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 }; // Buffer to store the returned UID
uint8_t uidLength; // Length of the UID (4 or 7 bytes depending on ISO14443A card type)
success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

long test = 0;
if (success){
  for (uint8_t i = 0; i < uidLength; i++)
  {
    // Serial.print("0x"); Serial.print(uid[i], HEX);
    test = test + uid[i];
    if (i<3) {
      test = test * 256;
    }
  }
  Serial.write("R");
  Serial.write(test);
} else {
  Serial.write("R0");
}

void task_done(){
  X_ADDRESS_TARGET = 0;
  Y_ADDRESS_TARGET = 0;
  Z_ADDRESS_TARGET = 0;
  F_PERCENT_SPEED = 100;
  OPCODE = 1000; // change OPCODE to 1000 when task is done
  // delay(2000);
  ready_command_lcd();
  Serial.write("X");
  Serial.print(CURRENT_X);
  Serial.write("Y");
  Serial.print(CURRENT_Y);
  Serial.write("Z");
  Serial.print(CURRENT_PAYLOAD_HEIGHT);
  Serial.write("A");
  Serial.print(CURRENT_THREAD_LEN_A);
  Serial.write("B");
  Serial.print(CURRENT_THREAD_LEN_B);
  Serial.write("C");
  Serial.print(CURRENT_THREAD_LEN_C);
  send_sensor();
  delay(1000);
  Serial.write("G"); // let matlab know when it is good to send another command.
```c
void set_pins()
{
  // LCD
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(8, OUTPUT);

  // STEPPERS
  pinMode(25, OUTPUT);
  pinMode(23, OUTPUT);
  pinMode(29, OUTPUT);
  pinMode(27, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(31, OUTPUT);
  pinMode(35, OUTPUT);
  pinMode(37, OUTPUT);
  pinMode(39, OUTPUT);
}
void setup()
{
  Commands.begin();
  lcd.begin(16, 2);
  lcd.print("<FP> ");
  lcd.setCursor(2, 1);
  lcd.print("Spydercam 18");
  delay(2000);
  lcd.clear();
  lcd.print(" BETA ");
  lcd.setCursor(0, 1);
  lcd.print("Arduino Firmware");
  delay(2000);
  init_values();
  delay(2000);
  set_pins();
  stepperA = AccelStepper(motorInterfaceType, stepPinA, dirPinA);
  stepperB = AccelStepper(motorInterfaceType, stepPinB, dirPinB);
  stepperC = AccelStepper(motorInterfaceType, stepPinC, dirPinC);
  digitalWrite(35, LOW);
  digitalWrite(37, LOW);
  digitalWrite(39, LOW);
  steppers.addStepper(stepperA);
  steppers.addStepper(stepperB);
  steppers.addStepper(stepperC);

  // sensor setup
  nfc.begin();
  lcd.clear();
  lcd.print(" Enabling NFC ");
  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();
    }
}
% ======== 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

% 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

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 ','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 '};
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

cipplet_to_gcode.ButtonPushedFcn = {@convertCallback, gcode_area, speed}; % Set callback for convert_to_gcode button. Down here so it can reference speed

% --- Execution Area --- %
incr_distance = uieditfield(gui,'Position',[800, 270, 50, 20]); % Incremental movement distance
incr_dlbl = uilabel(gui,'Text','Distance','Position',[750, 270, 60, 25]); % Label for distance
incr_speed = uieditfield(gui,'Position',[940, 270, 50, 20]); % Incremental movement speed
incr_slbl = uilabel(gui,'Text','Speed (IPM)','Position',[865, 270, 70, 25]); % Label for speed

% X, Y, and Z incremental movement buttons
yplus_btn = uibutton(gui,'push','Text','Y+', 'Position',[850, 530, 100, 100], 'ButtonPushedFcn',{@directionBtnCallback, "Y+", incr_distance, incr_speed});
xminus_btn = uibutton(gui,'push','Text','X-','Position',[740, 420, 100, 100], 'ButtonPushedFcn',{@directionBtnCallback, "X-", incr_distance, incr_speed});
xplus_btn = uibutton(gui,'push','Text','X+','Position',[960, 420, 100, 100], 'ButtonPushedFcn',{@directionBtnCallback, "X+", incr_distance, incr_speed});
yminus_btn = uibutton(gui,'push','Text','Y-','Position',[850, 310, 100, 100], 'ButtonPushedFcn',{@directionBtnCallback, "Y-", incr_distance, incr_speed});
zplus_btn = uibutton(gui,'push','Text','Z+','Position',[960, 530, 50, 50], 'ButtonPushedFcn',{@directionBtnCallback, "Z+", incr_distance, incr_speed});
zminus_btn = uibutton(gui,'push','Text','Z-','Position',[790, 360, 50, 50], 'ButtonPushedFcn',{@directionBtnCallback, "Z-", incr_distance, incr_speed});
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

% Buffer and sensor data control buttons
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
run_btn = uibutton(gui,'state','Text','Run','Position',[1020, 150, 40, 50], 'ValueChangedFcn', @runCallback, 'Tag','RUN'); % Start the gcode sequence
sensor_data_btn = uibutton(gui,'push','Text','Sensor Data','Position',[740, 210, 100, 20], 'ButtonPushedFcn',{@sensorViewCallback}); % Show the sensor data window
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], 'Tag','SER', '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
% 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
    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

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.
global gui;
global currentgcode;
children = get(gui,'Children');

% If dropdown changed from G00
if sum(prevcode == 'G00') == 3
    %Find the editfields/labels with the G00 tag and make them invisible.
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && (children(i).Tag == "G00")
            children(i).Visible = 'off';
            %Find the editfield and set its value to empty
            if length(children(i).Type) == 11
                children(i).Value = '';;
            end
        end
    end

    %Find the editfields/labels with the G01 or G00 tag and make them invisible. (G00 and G01 both have XYZ, G01 just also has F)
elseif sum(prevcode == 'G01') == 3
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && ((children(i).Tag == "G00") || (children(i).Tag == "G01"))
            children(i).Visible = 'off';
            %Find the editfield and set its value to empty
            if length(children(i).Type) == 11
                children(i).Value = '';
            end
        end
    end

    %Find the editfields/labels with the G04 tag and make them invisible
elseif sum(prevcode == 'G04') == 3
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && (children(i).Tag == "G04")
            children(i).Visible = 'off';
            %Find the editfield and set its value to empty
            if length(children(i).Type) == 11
                children(i).Value = '';
            end
        end
    end
else
end

%If the new code is G00, turn editfields/labels tagged G00 visible
if sum(code == 'G00') == 3
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && (children(i).Tag == "G00")
            children(i).Visible = 'on';
        end
    end
end
% Set the current G-Code to be filled with arguments
currentgcode = ['"G00" "" ""'];
% If the new code is G01, turn editfields/labels tagged G00 or G01 visible
elseif sum(code == 'G01') == 3
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && ((children(i).Tag == "G00") || (children(i).Tag == "G01"))
            children(i).Visible = 'on';
        end
    end
    % Set the current G-Code to be filled with arguments
    currentgcode = ['"G01" "" "" '"
end
% If the new code is G04, turn editfields/labels tagged G04 visible
elseif sum(code == 'G04') == 3
    for i = 1:length(children)
        if ~isempty(children(i).Tag) && (children(i).Tag == "G04")
            children(i).Visible = 'on';
        end
    end
    % Set the current G-Code to be filled with arguments
    currentgcode = ['"G04" "'];
else
    % Set the current G-Code (no arguments)
currentgcode = [convertCharsToStrings(code)];
end

function arg_boxes()
    % This function sets up the editfields and labels for argument input on
    % the G-Code Input. It puts all of the items in position, then makes
    % them all invisible
    global gui;
    % XYZ labels/editfields
    xl = uilabel(gui, 'Text', 'X', 'Position', [520 230 10 20], 'Tag', "G00");
    xe = uieditfield(gui, 'Position', [535 230 30 20], 'Tag', "G00", 'ValueChangedFcn', {@fieldChange, 'X', '0'});
    yl = uilabel(gui, 'Text', 'Y', 'Position', [580 230 10 20], 'Tag', "G00");
    ye = uieditfield(gui, 'Position', [595 230 30 20], 'Tag', "G00", 'ValueChangedFcn', {@fieldChange, 'Y', '0'});
    zl = uilabel(gui, 'Text', 'Z', 'Position', [640 230 10 20], 'Tag', "G00");
    ze = uieditfield(gui, 'Position', [655 230 30 20], 'Tag', "G00", 'ValueChangedFcn', {@fieldChange, 'Z', '0'});
    % F label/editfield
    fl = uilabel(gui, 'Text', 'F', 'Position', [640 190 10 20], 'Tag', "G01");
    fe = uieditfield(gui, 'Position', [655 190 30 20], 'Tag', "G01", 'ValueChangedFcn', {@fieldChange, 'F', '0'});
% PX dropdown/editfield
pxdd = uidropdown(gui,'Items',{'P','X'},'Position',[520 230 40 20],'Tag','"G04");
pxe = uieditfield(gui,'Position',[565 230 30 20],'Tag','"G04","ValueChangedFcn',{@fieldChange,'P',pxdd});
pxdd.ValueChangedFcn = { @pxChange , pxe };

% Turn all of the elements created in this function invisible
xl. Visible = 'off';
ze. Visible = 'off';
yl. Visible = 'off';
zl. Visible = 'off';
fl. Visible = 'off';
fe. Visible = 'off';
pxdd. Visible = 'off';
pxe. Visible = 'off';
end

function [gcode] = drawing2gcode(speed)
  % This function turns data from the user inputted drawing and turns it
  % into G-Code. It takes in a constant speed for the whole operation. It
  % orders the drawing lines by proximity and then turns that path into
  % G-Code

  % Setup
  F = speed;
global home;
global drawing_input;
sets = length(drawing_input);
if length(drawing_input{sets}) < 1
  sets = sets-1;
  end
  indexes = [];

  %Find the line set that starts closest to the origin
  lowest_dist = 100;
  for i = 1:1:sets
    temp = sqrt(sum((home(1:2) - drawing_input{i}(1,:)).^2));
    if temp < lowest_dist
      lowest_dist = temp;
      indexes(1) = i;
    end
  end
  %Add line set that starts closest to the origin to a new
  ordered array
  new_order{i} = drawing_input{indexes(1)};

  % Order the rest of of the line sets, next closest starting point after
  % current set endpoint.
while length(new_order) < sets
lowest_dist = 100;
for i = 1:1:sets
    if ~ismember(i, indexes)
        temp = sqrt(sum((new_order(end)(end,:)-
drawing_input{i}(1,:)).^2));
        if temp < lowest_dist
            lowest_dist = temp;
            indexes(length(indexes)+1) = i;
            new_order{length(new_order)+1} = drawing_input{
                i};
        end
    end
end

%Initialize gcode
gcode = {};
gcode{i} = sprintf(';');

% Turn each set into a set of G-Code Commands
for i = 1:1:sets
    %Go to location of first point and lower down into pen
    height
    gcode{length(gcode)+1} = ['G00 ', 'X' num2str(new_order{i}
    (1,1)) ' Y' num2str(new_order{i}(1,2)) ' Z3;'];
    gcode{length(gcode)+1} = ['G01 ', 'Z2 ' ' F' num2str(F) ';'];
    %Stay in pen heigh and go through the points in the set (draw the
    %line)
    for j = 2:1:length(new_order{i})
        gcode{length(gcode)+1} = ['G01 ', 'X' num2str(new_order{i}
        (j,1)) ' Y' num2str(new_order{i}(j,2)) ' F' num2str(F) ';'];
    end
    %Lift up out of pen height
    gcode{length(gcode)+1} = ['G01 ', 'Z3 ' ' F' num2str(F) ';'];
end

% == GUI ELEMENT CALLBACK FUNCTIONS == %
function axesCallback(obj,~)
    % This is the callback function for clicking on the Drawing
    % Axes.
    % It gets the mouse click location and sends it to the plotting
    % function.
    mouse_loc = obj.CurrentPoint;
    plot_input(obj,mouse_loc(1,1:2));
end

function drawBtnCallback(obj,~,obj2)
    % This is the callback function for clicking the "Draw Line"
    % Button.
    % It turns on the ability to draw on the axes.
global drawing_input;
global num_sets;

if obj.Value == 1 % If button is pressed
    obj2.HitTest = 'on'; % Turn axis clicking on
    obj.Text = 'End Line';
else % If button is not pressed
    (un-pressed? pulled?)
    obj2.HitTest = 'off'; % Turn axis clicking
    obj.Text = 'Draw Line';

    if ~isempty(drawing_input{num_sets+1}) % If something has
        been drawn in this set
        num_sets = num_sets + 1; % Add 1 to num_sets
        drawing_input{num_sets+1} = []; % Add another
        cell to user input
    end
end

function clearBtnCallback(~,~,obj2)
    % This is the callback function for the "Clear Drawing" button.
    % It just resets the user input/number of sets then plots an
    % graph.

    global drawing_input;
global num_sets;

    num_sets = 0;
drawing_input = cell(1);

    plot_input(obj2,[-1 -1]);
end

function GdownCallback(obj,event)
    %This is the callback function for the G-Code dropdown menu. It
    %grabs
    %the current and previous values, then sends them to the
disp_args
    %function

    val = obj.Value;
prevVal = event.PreviousValue;
disp_args(val,prevVal);
end

function enterCallback(~,~,text_area)
    %This is the callback function for the send gcode button. It
    %adds the
    %G-Code formed by the dropdown and argument inputs to the text
    %area.

    global currentgcode;
    index = length(text_area.Value)+1;
```matlab

text_area.Value{index} = sprintf('');
for i = 1:length(currentgcode)
    charcode = convertStringsToChars(currentgcode(i));
    if ~isempty(charcode)
        text_area.Value{index} = [text_area.Value{index} sprintf(charcode) sprintf(' ')];
    end
end
text_area.Value{index} = text_area.Value{index}(1:end-1);
text_area.Value{index} = [text_area.Value{index} sprintf(';')];
end

function fieldChange(obj,~,lbl,obj2)
    % This is the callback function for when the argument fields are updated. This function runs every time those are changed values.
    global currentgcode;
    % Depending on the argument label, add argument to current g code
    if lbl == 'X'
        currentgcode(2) = convertCharsToStrings([lbl obj.Value]);
    elseif lbl == 'Y'
        currentgcode(3) = convertCharsToStrings([lbl obj.Value]);
    elseif lbl == 'Z'
        currentgcode(4) = convertCharsToStrings([lbl obj.Value]);
    elseif lbl == 'F'
        currentgcode(5) = convertCharsToStrings([lbl obj.Value]);
    elseif lbl == 'P'
        currentgcode(2) = convertCharsToStrings([obj2.Value obj.Value]);
    else
        %
    end

function convertCallback(~,~,text_area,speed)
    % This is the callback function for the convert to G-Code button. It calls the drawing2gcode function on the drawing_input, then puts the % results in the text area.
    global drawing_input;
    if length(drawing_input{1}) > 0
        gcode = drawing2gcode(speed.Value);
        text_area.Value = gcode;
    end
end

function linebreakCallback(~,~,text_area)
    % This is the callback function for the line break button. It just % prints a ';' on the next line.
```
index = length(text_area.Value)+1;
text_area.Value{index} = sprintf(';');

function deleteLineCallback(~,~,text_area)
    % This is the callback function for the delete line button. It just
    % deletes the last line in the G-Code text box
    temp = {};
    if length(text_area.Value) > 1
        for i = 1:1:(length(text_area.Value)-1)
            temp{i} = text_area.Value{i};
        end
        text_area.Value = temp;
    end
end

function pxChange(obj,~,ef)
    % This is the callback for the P and X dropdown on the G04 argument
    % It updates the currentgcode value and also deletes the
    % editfield value if the drop down changes value
    global gui;
global currentgcode;
children = get(gui,'Children');
for i = 1:length(children)
    if length(children(i).Type) == 11
        children(i).Value = '';  
    end
end
    currentgcode(2) = convertCharsToStrings([obj.Value ef.Value]);
end

function clearGcodeCallback(~,~,text_area)
    % This is the callback function for the clear gcode button. It just
    % sets the text_area to be equal ';'
    text_area.Value = sprintf(';');
end

function send2bufferCallback(~,~,text_area)
    % This function is the callback for the send to buffer button. It adds
    % the lines from the text area to the buffer, then deletes the text
    % area.
    global gcode_buffer;
for i = 1:length(text_area.Value)
    gcode_buffer = [{text_area.Value{i}}; gcode_buffer];
end

text_area.Value = {'';
update_clear_btn();
update_bufferfig();
end

function directionBtnCallback(~,~ , dir , dist , speed)
% This is the callback function for the directional buttons. It
% takes a
% direction, distance, and speed, and then ti moves at that
% speed for
% that dist in that direction.

global gcode_buffer;

% If user didn’t set distance or speed, give them some default
values
if isempty(dist.Value)
dist.Value = '1';
end
if isempty(speed.Value)
speed.Value = '100';
end

% If direction = Y+,Y-,X+,X-,Z+,Z-
% add 'G91' to buffer (Set incremental movement)
% add 'G01' to buffer with distance and speed (Move in the
specified
% direction
% add 'G90' to buffer (Set absolute movement)
if dir == "Y+
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 Y' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
elseif dir == "X-
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 X-' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
elseif dir == "X+
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 X' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
elseif dir == "Y-
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 Y-' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
elseif dir == "Z+
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 Z' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
elseif dir == "Z-
    gcode_buffer = [{'G91';} ; gcode_buffer ];
gcode_buffer = [{['G01 Z-' dist.Value ' F' speed.Value ' ;
';} ; gcode_buffer ];
gcode_buffer = [{'G90 ;'} ; gcode_buffer ];
end
elseif dir == "Z-
    gcode_buffer = [{'G91 ;'}; gcode_buffer ];
    gcode_buffer = [[['G01 Z- ' dist.Value ' F' speed.Value ' ;']}; gcode_buffer ];
    gcode_buffer = [{'G90 ;'}; gcode_buffer ];
elseif dir == "home"
    gcode_buffer = [{'G28 ;'}; gcode_buffer ];
else
    end

    % Update the clear button because values have been added to the buffer
    update_clear_btn();
    update_bufferfig();
end

function bufferCallback(~,~)
    % This is the view gcode buffer button callback. It shows a figure which
    % contains a text area which contains the contents of the G-Code Buffer
    global bufferfig;
    global gcode_buffer;
    bufferfig = uitextarea('Position', [50 20 460, 380]);
    bufferfig.Parent.Name = 'G-Code Buffer';
    bufferfig.Value = flip(gcode_buffer);
end

function bufferClearCallback(~,~)
    % This function is the buffer clear button callback. It clears the buffer and updates the number on itself.
    global gcode_buffer;
    gcode_buffer = [];
    update_clear_btn();
    update_bufferfig();
end

function sensorViewCallback(~,~)
    % This is the callback function for the button that opens the sensor data view window. It opens the sensor data view window.
    global sensorViewFig;
    % Make the figure and the headings
    sensorViewFig = uifigure('Position', [200 100 600 500], 'Tag', 'svf');
    axes_label = uilabel(sensorViewFig, 'Text', 'Data Points', 'Position', [45 460 200 40], 'FontSize', 24);
    info_label = uilabel(sensorViewFig, 'Text', 'Sensor Data', 'Position', [390 460 200 40], 'FontSize', 24);
%Label and values for coordinates (data from Arduino)
coord_label = uilabel(sensorViewFig,'Text', 'Coordinates (Inches):', 'Position',[390 430 200 25],'FontSize',16);
coord_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390 410 200 25], 'Tag','COORD');

%Label and value for light sensor (data from Arduino)
lsv_label = uilabel(sensorViewFig,'Text', 'Light Sensor Value:', 'Position',[390 360 200 25], 'FontSize',16);
lsv_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390 340 200 25], 'Tag','LSV');

%Label and value for RFID (data from Arduino)
RFID_label = uilabel(sensorViewFig,'Text', 'RFID Value:', 'Position',[390 290 200 25], 'FontSize',16);
RFID_val = uilabel(sensorViewFig,'Text', ' ', 'Position',[390 270 200 25], 'Tag','RFID');

%Axes that will show data points
dataAxes = uiaxes(sensorViewFig,'Position', [20, 20, 340, 440], 'ButtonDownFcn',@dataAxesCallback); %Create Axes
dataAxes.XLim = [0 8.5]; %Set axes limits
dataAxes.YLim = [0 11];
dataAxes.PickableParts = 'all'; %Make axes clickable
dataAxes.HitTest = 'on'; %Make axes clickable
dataAxes.NextPlot = 'replacechildren';

%Scroll buttons and data point number. Scrolling increases/decreases
%data point number.
scroll_left = uibutton(sensorViewFig,'push','Text','<<','Position',[390, 50, 90, 22], 'ButtonPushedFcn',{ @leftScrollCallback,dataAxes});
scroll_right = uibutton(sensorViewFig,'push','Text','>>','Position',[490, 50, 90, 22], 'ButtonPushedFcn',{ @rightScrollCallback,dataAxes});
dpn_val = uilabel(sensorViewFig,'Text', 'Data Point Number:', 'Position',[390 75 200 25], 'Tag','DPN');

%Plot the sensor data on the axes
update_sensor_data(dataAxes);

function runCallback(obj,~)
    %This is the callback function for the run button. It starts running
    %whatever Gcode sequence is currently in the buffer by sending a start
    %signal to the Arduino. The Arduino responds with some initial
    %and sensor data, then MATLAB knows that it can send the first gcode
    %command
    global gcode_buffer;
global s;
end

function update_sensor_data(axes)
    %Plot sensor data
    %...
global serial_on;
global prgm_end;

% Program end set to 0 means the program is starting.
prgm_end = 0;
serial_on = 1;

% If there's stuff in the buffer and run has been turned on, send the
% start signal
if length(gcode_buffer) > 0 && obj.Value == 1
    % serial_on = 0;
    t = timer('TimerFcn', @timer_trash, 'StopFcn', {@send_gcode, s },'StartDelay',0.05);
    start(t);
else
    obj.Value = 0;
end

% If the button has been unpressed, press it again
if obj.Value == 0
    obj.Value = 1;
end

% If there's nothing in the buffer, the program can be ended by
% unpressing manually
if length(gcode_buffer) == 0 && obj.Value == 1
    obj.Value = 0;
end

function leftScrollCallback(~,~,dataAxes)
% This is the callback function for the left scroll button in
% the sensor view window. This decreases the datapoint number so that you
% can look % at the next data point on the list

global sensor_data;
sds = size(sensor_data);

% Find the data point colored blue
last = findobj(dataAxes.Children,'Color',[0 0 1]);
if size(last) > 0
    % get the data point number from the tags
    id = last.Tag;
    newid = str2num(id)-1; % decrease data point number (go to next point)
    % If the data point number is 0, go to the highest one
    if newid == 0
        newid = sds(1);
    end
    % Update the sensor view with the new selected data point
    update_sensor_view(num2str(newid), dataAxes);
else
    % If none had been selected previously, just select the
first one
    update_sensor_view('1',dataAxes);
end

function rightScrollCallback(~,~,dataAxes)
    global sensor_data;
    sds = size(sensor_data);
    last = findobj(dataAxes.Children,'Color',[0 0 1]);
    if size(last) > 0
        id = last.Tag;
        newid = str2num(id)+1;
        if newid == sds(1)+1
            newid = 1;
        end
        update_sensor_view(num2str(newid),dataAxes);
    else
        update_sensor_view('1',dataAxes);
    end
end

function dataPointCallback(self,~,dataAxes)
    id = self.Tag;
    update_sensor_view(id,dataAxes);
end

function dirSendCallback(~,~,dir_ser,ser_get)
    global s;
    % For every item in the serial send bar, send it on serial and delete
% the line from the text area.
for idx = 1:length(dir_ser.Value)
    disp(dir_ser.Value{idx});
    ser_get.Value{end+1} = '';
    ser_get.Value{end+1} = sprintf('Outgoing Serial');
    ser_get.Value{end+1} = sprintf(dir_ser.Value{idx});
    scroll(ser_get,'bottom');

    % Only send if its not an empty line.
    if ~isempty(dir_ser.Value{idx})
        writeln(s,dir_ser.Value{idx});
        dir_ser.Value{idx} = '';
    end
end

function serGetCallback(self,~)
    %Scroll the serial monitor to the bottom
    scroll(self,'bottom');
end

function dataAxesCallback(~,~)
    %This callback function does nothing
end

% =========== UPDATE FUNCTIONS ===========
function update_clear_btn()
    % This function updates the display on the buffer clear button
    % tells you how many lines are in the buffer
    global gui;
    global gcode_buffer;
    children = get(gui,'Children');
    % Find the button by Tag and then set the number in the text to include
    % new size of gcode_buffer
    for i = 1:length(children)
        if children(i).Tag == 'C'
            children(i).Text = ['Clear Buffer (' int2str(length(gcode_buffer)) ')'];
        end
    end

    if length(gcode_buffer) == 0
        run_btn = findobj(gui.Children,'Tag','RUN');
        run_btn.Value = 0;
    end
end

function update_bufferfig()
    %This function updates the contents of the text box in the external
    %figure which shows the GCode buffer.
global bufferfig;
global gcode_buffer;

% If the gcode buffer has stuff in it and the figure exists, set the new value of the text box
if length(gcode_buffer) > 0 && sum(size(findobj('Value',bufferfig))) > 0
    bufferfig.Value = flip(gcode_buffer);
end

function update_sensor_data(dataAxes)
    % This function updates the plot in the sensor view window. It plots all of the newest sensor data on the axes.
    global sensor_data;

    % Set the axes to replace old plots with new ones, then plot a garbage point
    dataAxes.NextPlot = 'replacechildren';
    plot(dataAxes,-1,-1,'ro');

    % Set axes to ADD all new plots
    dataAxes.NextPlot = 'add';

    sd = size(sensor_data);

    % Plot all of the points
    for i = 1:1:sd(1)
        plot(dataAxes,sensor_data(i,1),sensor_data(i,2),'ro','Tag',num2str(i),'ButtonDownFcn',{@dataPointCallback,dataAxes});
    end

end

function update_sensor_view(id,dataAxes)
    % This function updates the sensor view window given a newly selected data point. The data point with data point number 'id' is turned blue and its data is shown on the right hand side of the window.
    global sensorViewFig;
global sensor_data;

    % Find the point that’s colored blue
    last = findobj(dataAxes.Children,'Color',[0 0 1]);

    % If there was a blue point, turn it back to red
    if size(last) > 0
        last.Color = [1 0 0];
        last.MarkerSize = 6;
    end
% Get the point with data point number id and turn it blue
curr = findobj(dataAxes.Children,'Tag',id);
curr.Color = [0 0 1];
curr.MarkerSize = 10;

nid = str2num(id);

% Grab all of the data display labels
coord_lbl = findobj(sensorViewFig.Children,'Tag','COORD');
lsv_lbl = findobj(sensorViewFig.Children,'Tag','LSV');
rfid_lbl = findobj(sensorViewFig.Children,'Tag','RFID');
dpn_lbl = findobj(sensorViewFig.Children,'Tag','DPN');

% Get all of the values to display from the sensor data (depends on id)
x = num2str(sensor_data(nid,1));
y = num2str(sensor_data(nid,2));
z = num2str(sensor_data(nid,3));
lsv = num2str(sensor_data(nid,7));
rfid = num2str(sensor_data(nid,8));

% Change the text of the labels to match the data gathered from sensor_data
coord_lbl.Text = ['X: ' x ' Y: ' y ' Z: ' z];
lsv_lbl.Text = lsv;
rfid_lbl.Text = rfid;
dpn_lbl.Text = ['Data Point Number: ' id];

end

% ======================== %
% === SERIAL FUNCTIONS === %

function readSerialData(src,~)
    global sensor_data;
    global gui;
    global sensorViewFig;
    global serial_on;
    values = zeros(1,8);

    % If there are bytes available and serial is enabled, grab all of the new data.
    serial_on = 1;
    if src.NumBytesAvailable > 0 && serial_on == 1
        values = getSerial(src);
        sensor_data = [sensor_data; values];
    
    % If the sensor view window is open, update the axes.
    if size(findobj(sensorViewFig,'Type','figure')) > 0
        dataAxes = findobj(sensorViewFig.Children,'Type','axes');
        if size(dataAxes) > 0
update_sensor_data(dataAxes);
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: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);
            end
        end
    end
end
start(t);
end
elseif data(i) == '#'
    while character ~= 'G'
        character = read(src,1,"char");
    end
    character = read(src,1,"char");
j=-1;
end

%If there's a new letter, put all of the previous numbers into the sensor values
if j ~= lastj && lastj ~= -1
    values(1,lastj) = str2double(current(2:end));
current = ' ';
end

%If j is unchanged (same letter), keep adding the information
if j ~= -1
    current = [current data(i)];
end

lastj = j;
end

ser_get = findobj(gui,'Tag','SER');

if sum(values(1:3)) ~= 0
    disp("Incoming Location Data:");
    disp(values(1:3));
    ser_get.Value{end+1} = sprintf('Incoming Location Data:
Y: ' num2str(values(2)) ' Z: ' num2str(values(3))]);
    scroll(ser_get,'bottom');
end

if sum(values(4:6)) ~= 0
    disp("Incoming Thread Length Data:");
    disp(values(4:6));
    ser_get.Value{end+1} = sprintf('Incoming Thread Length Data :'
B: ' num2str(values(5)) ' C: ' num2str(values(6))]);
    scroll(ser_get,'bottom');
end

function send_gcode(~,~,src)
    %This function sends 1 line of gcode through serial to the Arduino.
    %Because of the nature of the serial read callback, this function has
    %to be set up as a callback function for a timer.
    global gcode_buffer;
global serial_on;
global gui;
global prgm_end;

% Grab the serial monitor
ser_get = findobj(gui, 'Tag', 'SER');

serial_on = 0;

%If the gcode buffer has stuff in it and there's nothing more to read, %send the stuff.
if length(gcode_buffer) > 0 && src.NumBytesAvailable == 0
    % get rid of all of the empty lines or line breaks
    while (sum(gcode_buffer{end} == ';') == 1 && length(gcode_buffer{end})) == 1) || length(gcode_buffer{end}) == 0
        gcode_buffer{end} = [];
    end
    writeline(src, gcode_buffer{end});
disp("Outgoing GCode:");
disp(gcode_buffer{end});
% Print gcode to serial monitor
    ser_get.Value{end+1} = '';
    ser_get.Value{end+1} = sprintf('Outgoing Gcode');
    ser_get.Value{end+1} = sprintf(gcode_buffer{end});
    scroll(ser_get, 'bottom');
    gcode_buffer{end} = [];
    update_clear_btn();
end

% Only turn the serial back on if the program is not ended
if prgm_end == 0
    serial_on = 1;
end

update_bufferfig();

function timer_trash(~,~)

function end_program(~,~)
% This function is run when the current gcode sequence has run its course. It deactivates serial, deactivates the run button, and sets the program end flag to true.

global gui;
global serial_on;
global prgm_end;

run_btn = findobj(gui.Children,'Tag','RUN');
run_btn.Value = 0;
serial_on = 0;
prgm_end = 1;
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.

![Diagram showing calculation of thread lengths](image)

**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.