SpyderCam Team 23 Developer Guide

Sawyer Brundage, Camden Robustelli, Mikhail Burlachenko, Ali Alfadala

March 2021

1 System Overview

This SpyderCam can maneuver around a standard 8.5" by 11" sheet of paper. It consists of a three stepper motor system that suspends a payload over the specified area. The stepper motors are positioned in a triangle shape and are controlled by a lookup table. There are two ways to control the SpyderCam Joystick or typing in a G-Code command to change position. The joystick reads analog x and y inputs that allow you to move around the piece of paper. The straight line speed of the SpyderCam is 4 inches per second.

2 Electrical Specifications

This system runs on battery pack power supplies that are 12V each. For the motors we have two of them combined making a 24V power supply feeding the three motors. The arduino is hooked up to the computer to run our matlab code so it is powered by USB. For the NEMA-17 stepper motors the operating temperature is -10 to 40 degrees C. The nominal current is 1.2A at 4V and the maximum voltage you can put into the stepper motors is 35V. The L298N motor drivers have a storage temperature of between -20 to +135 degrees C, and outputs a maximum power of 25W. In this case, where the driving voltage is 24 V, the 5V jumper cap must be disabled and an outside 5V source must be used (the Arduino, in this case).

3 User Guide

3.1 Setting Up System

Have the MATLAB GUI and code up and running as well as the Arduino code imported and running in the Arduino Uno. Flip the enable switch to the "on" position to power the motors and motor drivers. With the payload hovering over the center of the 8.5" by 11" piece of paper, click the "calibrate" button on the GUI and begin operation.

3.2 Operating System

Once the system is setup you use the GUI in MATLAB to pick whether you want to control with a joystick or G-Code mode. This is in a drop down box menu labeled input mode. On the GUI there is a spot to put in G-Code commands and the location that the payload is at. Pressing the calibrate button will set the SpyderCam's current location to (0, 0). The command line supports G0, G1, G20, G21, G90, G91, M2, and M6 commands. The desired units can also be switched between inches and millimeters using the appropriate dropdown menu.

4 Design Artifacts

4.1 Top Level Block Diagram

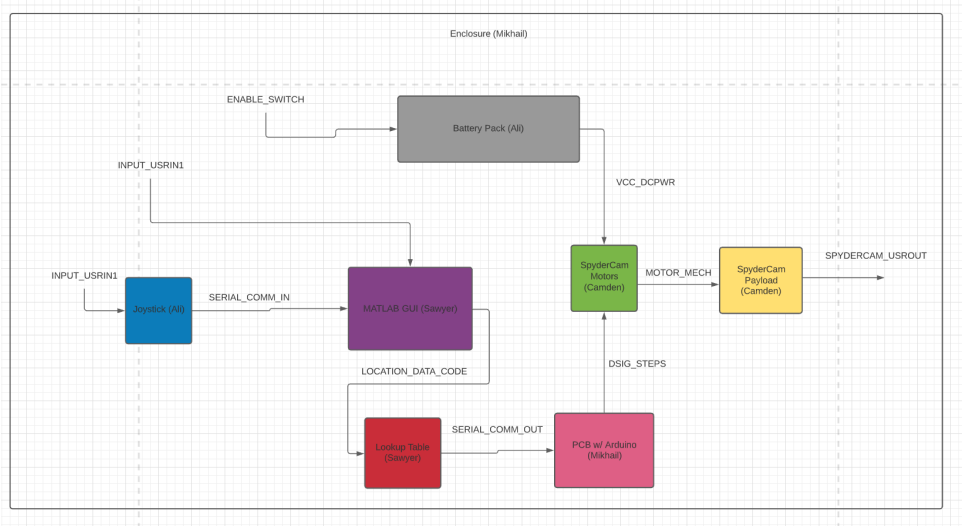

Figure 1: Block Diagram

4.2 Interface Definitions

INPUT_USERIN1:

Joystick movements from the user that are read by the Arduino
Supports four directions: up, down, left, right

INPUT_USERIN2:

Keyboard and Mouse Inputs from the user.
Interacts with the GUI interface through button presses and text input.

ENABLE_SWITCH:

Simple on/off switch for the battery case

Switched on: battery outputs 24 V

Switched off: block is deactivated (0 V of output voltage)

SERIAL_COMM_IN:

Data rate: 9600 baud

Logic Level: +5V to 0V

UART TTL Serial Communication with Arduino Nano's ATmega168 microcontroller

Arduino prints strings to serial, which are read by MATLAB

LOCATION_DATA_CODE:

XY coordinates of the payload's current location and destination

Coordinates are converted to indices for the table

SERIAL_COMM_OUT:

Data rate: 9600 baud

Logic Level: +5V to 0V

UART TTL Serial Communication with Arduino Nano's ATmega168 microcontroller

MATLAB prints string, doubles, and integers to serial, which are read by Arduino

Outputs number of steps, feed rates, etc.

DSIG_STEPS:

Logic Level: +5 V to 0 V

4 pin connections between each motor driver and Arduino

Generated using functions from AccelStepper.h library

MOTOR_MECH:

Stepper motors rotate through the designated number of steps

200 steps per revolution

Supports up to 600 rpm

Wind or unwinds a thread of fishing line attached to payload

SPYDERCAM_USROUT:

Payload moves according to user's input from G-Code or Joystick

Three motors move the payload around the designated area

Can have a drawing utensil, scanner, or camera attached to the payload

VCC_DCPWR:

Maximum Voltage: 24 V

Minimum Voltage: 0 V

Powers each stepper motor with 1.2 A of current

4.3 MATLAB GUI

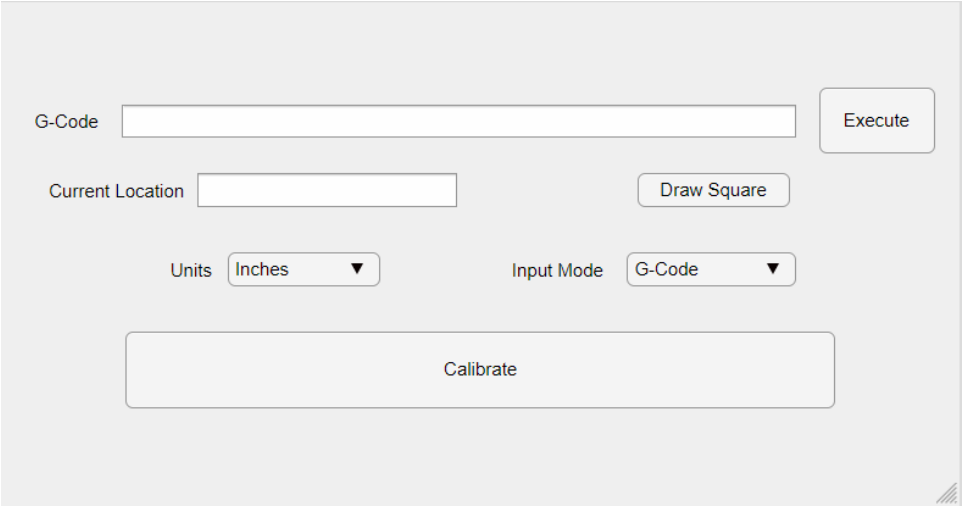

Figure 2: MATLAB GUI

The MATLAB GUI allows the user to input G-Code commands for the SpyderCam. It specifically allows for G0, G1, G90, G91, M2, and M6. G20 and G21 are implemented using the "Units" dropdown menu to change between millimeters and inches. The interface allows the user to change between millimeters and inches, G-Code control and joystick functionality, a subroutine that draws a square, and a text field that shows the SpyderCam's current xy coordinates. The calibrate button also sets the current location to (0, 0). When a command forcing the SpyderCam to move is executed (or the joystick is moved), the program sends data regarding the SpyderCam's current location and destination to the Lookup Table in order to obtain the number of steps needed for each motor. The code for the GUI is located at the end of this document under "Code Used".

4.4 Lookup Table

The Lookup Table is the "black box" that takes in location coordinates from the MATLAB GUI and converts it into the number of steps each motor must rotate. In order to achieve this, a function is called as the GUI launches and populates the table beforehand to minimize delay. The table calculates the length between each motor and a finite number of points on the 8.5" by 11" piece of paper, which are spaced apart by 1/16 of an inch. This length is then

converted into the number of steps from the initial point (0, 0). The results of the table from the SpyderCam's current coordinates are subtracted from the results of the destination, and the final result is written to the Arduino. The Lookup Table code is included at the end of the document under "Code Used".

4.5 Motor Spool

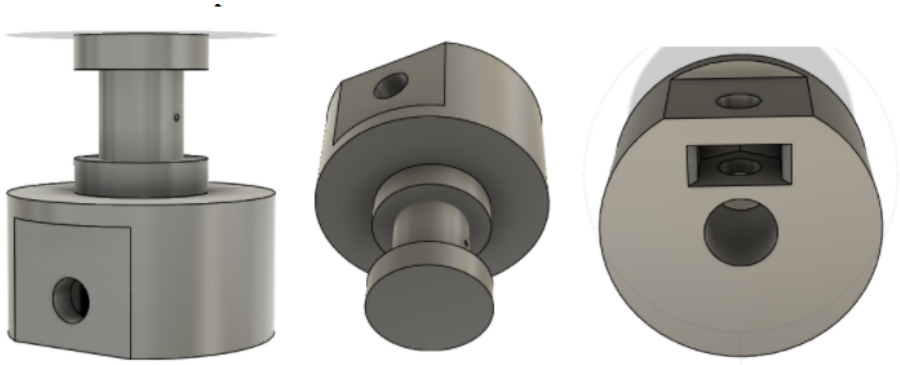

Figure 3: Spools for Stepper Motors

This spool when attached to the motor allows us to easily connect are cables from the motor to the SpyderCam. It is attached to the motor with a M5 hex nut with a 8mm M3 grub screw onto the shaft of the motor. The radius of where the cable for the SpyderCam is spooled up is $\frac{1}{4}$ of an inch. The stepper motor can run up to 600 rpm. At 500 rpm it can move at about 7.6 inches per second which is over the required speed. It has a hole in the middle of the spool to thread the cable into and tie off so the cable starts in the center. It has some wiggle room between the edges of the spool so the cable doesn't have to be exactly straight.

4.6 Payload

The payload designed for the SpyderCam system is in a triangular shape to easily anchor from the points to the stepper motors using a cable and cut down on weight. The cables run through the edge of the payload into the center to try and help reduce wobble. The cylinder in the center is designed to hold a laser upright so it can shine down onto the surface. This allows you to track the movement of the payload. Each side of the payload is 60 mm and the thickness of the payload is 26 mm to contain the laser.

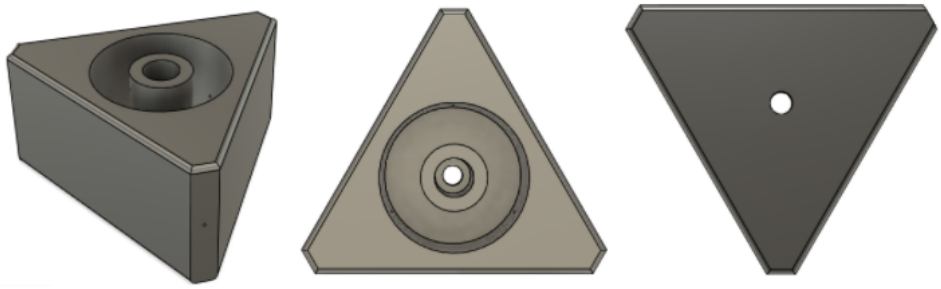

Figure 4: SpyderCam Payload

4.7 Arduino Code

The Arduino code is essentially split into two parts: handling joystick inputs to send to MATLAB and taking in data *from* MATLAB to then talk to the motors. When the joystick reads an "up", "down", "left", or "right" from analog signals, it will send corresponding 'U', 'D', 'L', or 'R' characters to MATLAB *if* it is in joystick mode. The Arduino also reads from the serial port to see if it should switch modes or tell the motors to move. If MATLAB sends the number of steps for each motor to Arduino, as well as the desired rpm, then Arduino will use AccelStepper.h library functions to move the motors synchronously according to the information. The Arduino code discussed is located at the end of this document under "Code Used."

4.8 Enclosure

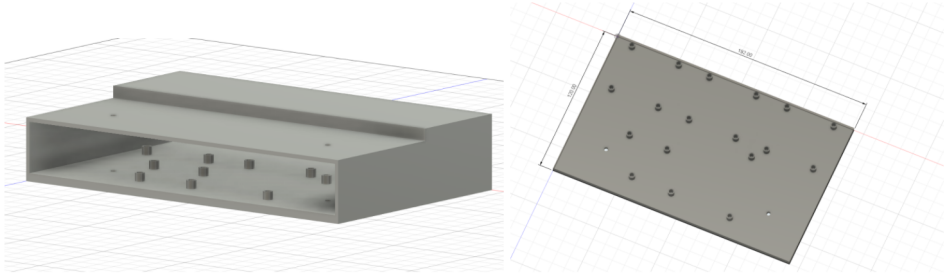

Figure 5: 3D Model of the PCB Enclosure

Enclosure is designed to cover three L298N motor drivers, Arduino Uno, and PCB. M2 heat-set inserts 6mm M2 screws are used to attach modules to the enclosure. Modules are attached to the bottom part (on the right). The cover part is attached to the bottom part with M2.5 25mm+6mm male-female hex standoffs and M2.5 hex nuts. The front of the enclosure is left

open for motor and power wires, and Arduino USB cable. The bottom part is 120mm×192mm×4mm. The cover is 27mm above the Arduino and PCB and 33mm above motor drivers.

5 PCB Information

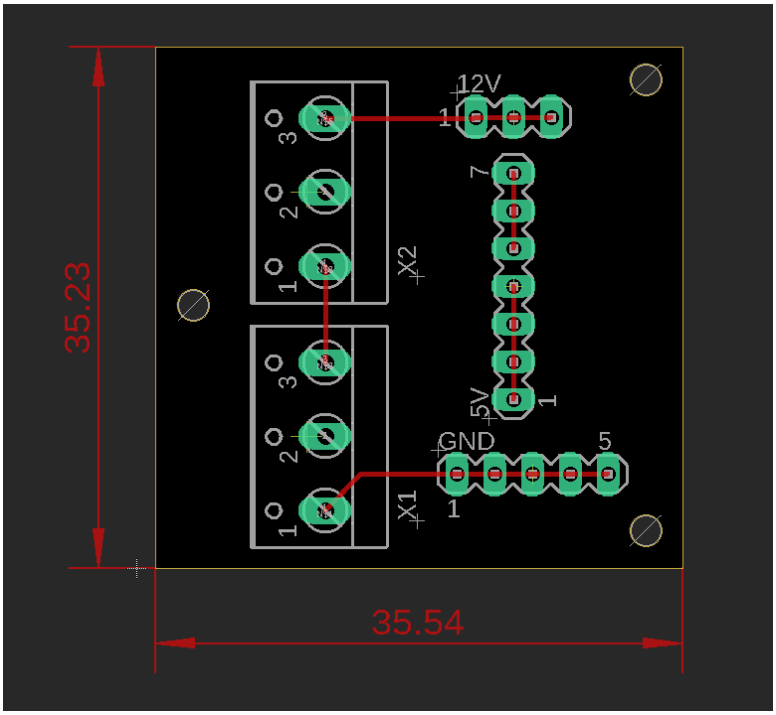

Figure 6: Diagram of the PCB (Units in mm)

The PCB is designed to power motors and a laser pointer. There are two 3-pin terminal blocks since two 12V battery packs are used to power the Spydercam. 12V pins and GND are connected to motor driver input. In case L298N motor drivers are powered with more than 12V, they require additional 5V input for the switching logic circuitry inside L298N. The 5V Arduino pin is connected to pin 7 on PCB and Arduino GND is connected to PCB GND. Pins 1, 2, and 3 provide additional 5V to motor drivers. If the power supply is more than 12V, a jumper should be used between pins 4 and 5 for 5V PCB. Pin 6 on the 5V PCB is connected to a laser pointer as well as one of the GND pins.

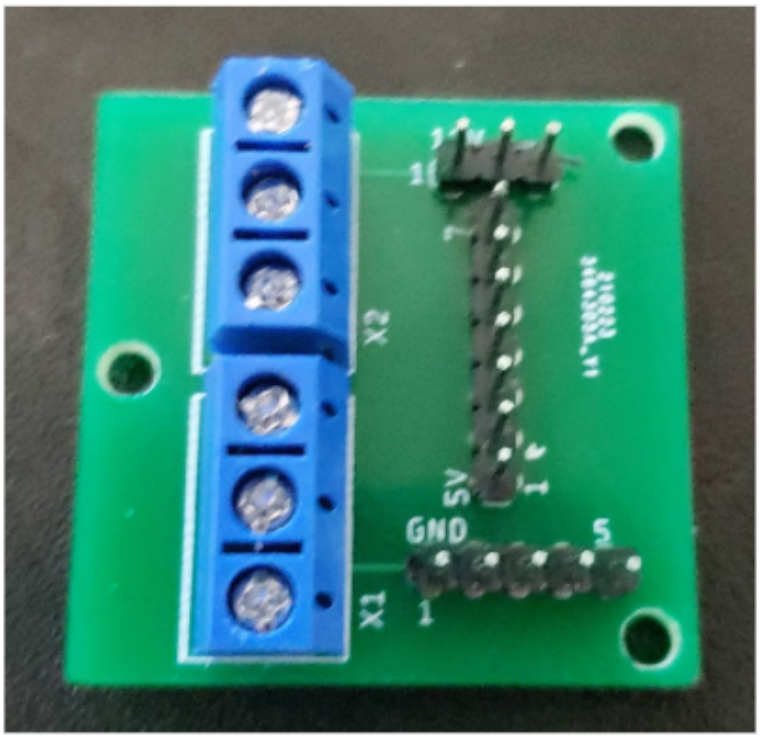

Figure 7: Complete PCB

6 Part Information

Figure 7 shows a list of parts used for this project. In this case, the team used and Arduino UNO as a microcontroller. However, an Arduino Nano would work as well.

Part:	Quantity:	Price:
Particle Board ⅝" x 2' x 4'	1	$10.05
Cedar Square End 2" x 2" x 48"	1	$3.98
Motor Controller L298N	1 (pack of 4)	$9.99
NEMA-17 Stepper Motor	1 (pack of 3)	$25.99
Spiderwire Fishing Line	1	$16.79
AA 12v Battery Power Supply	1 (pack of 2)	$8
AA Batteries	1 (pack of 24)	$16.24
Motor Mounting Brackets	1 (pack of 5)	$13.99
Laser For Payload	1	$13.18
Hex Nut M5	3	$.60
Grub Screw M3	3	$.65
3D Printed Payload and Motor Spools	1	$12
Printed PCB	1	$19.80
PCB Enclosure	1	$1.20
Heat-set Insert M2	18	$1.43
Screw M2	18	$1.26
3-pin Terminal Block	2	$2
Hex Standoff M2.5	2	-
Screw M2.5*6	2	-
Hex M2.5	2	-
Arduino Uno	1	-
Arduino-Compatible Joystick	1	-
Total		$152.46

Figure 8: List of Parts Used

7 Code Used

7.1 MATLAB GUI

```
classdef SpyderCamGUI < MATLAB.apps.AppBase
```

```
% Properties that correspond to app components
properties (Access = public)
    UIFigure                        MATLAB.ui.Figure
    InputModeDropDownLabel          MATLAB.ui.control.Label
    InputModeDropDown               MATLAB.ui.control.DropDown
    ExecuteButton                   MATLAB.ui.control.Button
    UnitsDropDownLabel              MATLAB.ui.control.Label
    UnitsDropDown                   MATLAB.ui.control.DropDown
    CalibrateButton                 MATLAB.ui.control.Button
    GCodeEditFieldLabel             MATLAB.ui.control.Label
    GCodeEditField                  MATLAB.ui.control.EditField
    CurrentLocationEditFieldLabel   MATLAB.ui.control.Label
    CurrentLocationEditField        MATLAB.ui.control.EditField
    DrawSquareButton                MATLAB.ui.control.Button
end


properties (Access = private)
    xpos = 0;   % Tracks coordinates of SpyderCam
    ypos = 0;
    device = serialport("COM3", 9600);      % Sets up communication with Arduino
    mode = 0;   % Value switches between 0 and 1, depending if in G90 or G91 mode
    LookupTable = FurnishLookupTable();     % Creates a Lookup Table struct array that
                                            % determines number of steps for each motor
    input_method = "G-Code";
end

methods (Access = private)



    function results = WriteSteps(app, xnew, ynew, feed)
            i = round(((xnew + 5.5)/11)*704) + 1;
            j = round(((ynew + 4.25)/8.5)*544) + 1;
            k = round(((app.xpos + 5.5)/11)*704) + 1;
            l = round(((app.ypos + 4.25)/8.5)*544) + 1; % Converts coordinates to
                                                        % index values for Lookup Table

            steps1 = compose("%d\n", app.LookupTable(i, j).motor1 -
            app.LookupTable(k, l).motor1);
            steps2 = compose("%d\n", app.LookupTable(i, j).motor2 -
            app.LookupTable(k, l).motor2);
            steps3 = compose("%d\n", app.LookupTable(i, j).motor3 -
            app.LookupTable(k, l).motor3);
            feedstr = compose("%d\n", feed);
```

```
            %Writes step values and feed rate to serial
            write(app.device, steps1, "string");
            write(app.device, steps2, "string");
            write(app.device, steps3, "string");
            write(app.device, feedstr, "string");

            app.xpos = xnew;
            app.ypos = ynew;
    end
end


% Callbacks that handle component events
methods (Access = private)

    % Button pushed function: ExecuteButton
    function ExecuteButtonPushed(app, event)
        [token1, remain1] = strtok(app.GCodeEditField.Value);

        if (token1 == "G90")
            app.mode = 0;       % Absolute Mode

        elseif (token1 == "G91")
            app.mode = 1;       % Incremental Mode

        elseif (token1 == "G0") % Rapid Movement: no specified feed value
            [~, remain2] = strtok(remain1, "X");
            [token2, remain3] = strtok(remain2);
            [~, remain4] = strtok(remain3, "Y");
            token3 = erase(strtok(remain4), "Y");
            token2 = erase(token2, "X");        % Grabs X and Y values

            switch (app.mode)       % Absolute or Incremental
                case 0
                    if (app.UnitsDropDown.Value == "Inches")
                        xnew = str2double(token2);
                        ynew = str2double(token3);

                    else
                        xnew = str2double(token2) / 25.4;   % Converts input to
                                                            % inches from mm
                        ynew = str2double(token3) / 25.4;
                    end

                case 1
```

```
            if (app.UnitsDropDown.Value == "Inches")
                xnew = str2double(token2) + app.xpos;
                ynew = str2double(token3) + app.ypos;

            else
                xnew = str2double(token2) / 25.4 + app.xpos;    % Converts
                                                                % to inches
                ynew = str2double(token3) / 25.4 + app.ypos;
            end
        end

        if (xnew <= 5.5 && xnew >= -5.5 && ynew <= 4.25 && ynew >= -4.25) % Checks
        if values are valid
            app.WriteSteps(xnew, ynew, 500);       % Default Feed value is 500
        end

    elseif (token1 == "G1")     % Controlled Movement: Feed value is
                                % specified by user
        [~, remain2] = strtok(remain1, "X");
        [token2, remain3] = strtok(remain2);
        [~, remain4] = strtok(remain3, "Y");
        [token3, remain5] = strtok(remain4);
        [~, remain6] = strtok(remain5, "F");
        token4 = erase(strtok(remain6), "F");
        token2 = erase(token2, "X");
        token3 = erase(token3, "Y");        % Grabs X, Y, and F values
                                            % from edit field

        switch (app.mode)
            case 0
                if (app.UnitsDropDown.Value == "Inches")
                    xnew = str2double(token2);
                    ynew = str2double(token3);

                else
                    xnew = str2double(token2) / 25.4;       % Converts to inches
                    ynew = str2double(token3) / 25.4;
                end

            case 1
                if (app.UnitsDropDown.Value == "Inches")
                    xnew = str2double(token2) + app.xpos;
                    ynew = str2double(token3) + app.ypos;

                else
                    xnew = str2double(token2) / 25.4 + app.xpos;    % Converts to
```

```
                                                                    % inches
                    ynew = str2double(token3) / 25.4 + app.ypos;
                end
            end

            feed = str2num(token4);
            if (xnew <= 5.5 && xnew >= -5.5 && ynew <= 4.25 && ynew >= -4.25 &&
            feed <= 600 && feed > 0)    % Checks if values are valid
                app.WriteSteps(xnew, ynew, feed);
            end

        elseif (token1 == "M2") % Terminates the program
            app.delete();

        elseif (token1 == "M7")  % Returns SpyderCam to center position
            app.WriteSteps(0, 0, 500);
        end


        if (token1 ~= "M2")
            if (app.UnitsDropDown.Value == "Inches")
                app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f", app.xpos,
                app.ypos);    % Shows current coordinates in inches

            else
                app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f", app.xpos *
                25.4, app.ypos * 25.4);     % ... or in mm
            end

            app.GCodeEditField.Value = sprintf("");     % Clears G-Code Input Field
        end

    end

    % Button pushed function: CalibrateButton
    function CalibrateButtonPushed(app, event)
        app.xpos = 0;
        app.ypos = 0;
        app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f", app.xpos, app.ypos);
    end

    % Value changed function: InputModeDropDown
    function InputModeDropDownValueChanged(app, event)
        flush(app.device);

        if (app.InputModeDropDown.Value == "G-Code")
```

```
            write(app.device, "G-Code", "string");
            app.input_method = "G-Code";            % Tells Arduino to expect
                                                    % step and feed values

        else
            write(app.device, "Joystick", "string");
            app.input_method = "Joystick";          % Tells Arduino to print values
                                                    % corresponding to joystick inputs
        end

        while (app.input_method == "Joystick")
            if (app.device.NumBytesAvailable > 0)
                val = read(app.device, 1, "string");
                flush(app.device);

                if (val == "U" && app.ypos <= 4.1875)   % Moves SpyderCam up 1/16
                    app.WriteSteps(app.xpos, app.ypos + 0.0625, 500);

                elseif (val == "D" && app.ypos >= -4.1875)  % Moves SpyderCam down 1/16
                    app.WriteSteps(app.xpos, app.ypos - 0.0625, 500);

                elseif (val == "R" && app.xpos <= 5.4375)   % Moves SpyderCam right
                    app.WriteSteps(app.xpos + 0.0625, app.ypos, 500);

                elseif (val == "L" && app.xpos >= -5.4375)  % Moves SpyderCam left
                                                            % 1/16 inches
                    app.WriteSteps(app.xpos - 0.0625, app.ypos, 500);

                end

                if (app.UnitsDropDown.Value == "Inches")
                    app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f",
                    app.xpos, app.ypos);

                else
                    app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f",
                    app.xpos * 25.4, app.ypos * 25.4);
                end

            end

            pause(0.001);   % Checks for a potential change in input mode,
                            % or some other interrupt
        end
    end
```

```
    % Value changed function: UnitsDropDown
    function UnitsDropDownValueChanged(app, event)

        % Updates the current coordinate values based on units
        if (app.UnitsDropDown.Value == "Inches")
            app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f",
            app.xpos, app.ypos);

        else
            app.CurrentLocationEditField.Value = sprintf("X:%f   Y:%f",
            app.xpos * 25.4, app.ypos * 25.4);
        end
    end

    % Drop down opening function: InputModeDropDown
    function InputModeDropDownOpening(app, event)
        app.input_method = "";
    end

    % Button pushed function: DrawSquareButton
    function DrawSquareButtonPushed(app, event)
        app.WriteSteps(0, 0, 500);

        app.WriteSteps(2, 0, 500);

        app.WriteSteps(2, 2, 500);

        app.WriteSteps(-2, 2, 500);

        app.WriteSteps(-2, -2, 500);

        app.WriteSteps(2, -2, 500);

        app.WriteSteps(2, 0, 500);
    end
end

% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)

        % Create UIFigure and hide until all components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Position = [100 100 629 330];
```

```
app.UIFigure.Name = 'MATLAB App';

% Create InputModeDropDownLabel
app.InputModeDropDownLabel = uilabel(app.UIFigure);
app.InputModeDropDownLabel.HorizontalAlignment = 'right';
app.InputModeDropDownLabel.Position = [331 145 66 22];
app.InputModeDropDownLabel.Text = 'Input Mode';

% Create InputModeDropDown
app.InputModeDropDown = uidropdown(app.UIFigure);
app.InputModeDropDown.Items = {'G-Code', 'Joystick', ''};
app.InputModeDropDown.DropDownOpeningFcn = createCallbackFcn(app,
@InputModeDropDownOpening, true);
app.InputModeDropDown.ValueChangedFcn = createCallbackFcn(app,
@InputModeDropDownValueChanged, true);
app.InputModeDropDown.Position = [412 145 111 22];
app.InputModeDropDown.Value = 'G-Code';

% Create ExecuteButton
app.ExecuteButton = uibutton(app.UIFigure, 'push');
app.ExecuteButton.ButtonPushedFcn = createCallbackFcn(app,
@ExecuteButtonPushed, true);
app.ExecuteButton.Position = [538 232 76 43];
app.ExecuteButton.Text = 'Execute';

% Create UnitsDropDownLabel
app.UnitsDropDownLabel = uilabel(app.UIFigure);
app.UnitsDropDownLabel.HorizontalAlignment = 'right';
app.UnitsDropDownLabel.Position = [107 145 33 22];
app.UnitsDropDownLabel.Text = 'Units';

% Create UnitsDropDown
app.UnitsDropDown = uidropdown(app.UIFigure);
app.UnitsDropDown.Items = {'Inches', 'Millimeters'};
app.UnitsDropDown.ValueChangedFcn = createCallbackFcn(app,
@UnitsDropDownValueChanged, true);
app.UnitsDropDown.Position = [150 145 100 22];
app.UnitsDropDown.Value = 'Inches';

% Create CalibrateButton
app.CalibrateButton = uibutton(app.UIFigure, 'push');
app.CalibrateButton.ButtonPushedFcn = createCallbackFcn(app,
@CalibrateButtonPushed, true);
app.CalibrateButton.Position = [83 65 466 50];
app.CalibrateButton.Text = 'Calibrate';
```

```
            % Create GCodeEditFieldLabel
            app.GCodeEditFieldLabel = uilabel(app.UIFigure);
            app.GCodeEditFieldLabel.HorizontalAlignment = 'right';
            app.GCodeEditFieldLabel.Position = [19 242 47 22];
            app.GCodeEditFieldLabel.Text = 'G-Code';

            % Create GCodeEditField
            app.GCodeEditField = uieditfield(app.UIFigure, 'text');
            app.GCodeEditField.Position = [81 242 442 22];

            % Create CurrentLocationEditFieldLabel
            app.CurrentLocationEditFieldLabel = uilabel(app.UIFigure);
            app.CurrentLocationEditFieldLabel.HorizontalAlignment = 'right';
            app.CurrentLocationEditFieldLabel.Position = [28 197 94 22];
            app.CurrentLocationEditFieldLabel.Text = 'Current Location';

            % Create CurrentLocationEditField
            app.CurrentLocationEditField = uieditfield(app.UIFigure, 'text');
            app.CurrentLocationEditField.Position = [130 197 171 22];

            % Create DrawSquareButton
            app.DrawSquareButton = uibutton(app.UIFigure, 'push');
            app.DrawSquareButton.ButtonPushedFcn = createCallbackFcn(app,
            @DrawSquareButtonPushed, true);
            app.DrawSquareButton.Position = [419 197 100 22];
            app.DrawSquareButton.Text = 'Draw Square';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = SpyderCamGUI

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
```

```
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

7.2 Lookup Table (MATLAB)

```
function [LookupTable] = FurnishLookupTable ()
    LfromP = 5;     % Initial length from paper for each pylon

    % Calculates initial thread length for each motor
    Init1 = LfromP + 5.5;
    Init2 = sqrt((LfromP*0.707 + 5.5)^2 + (LfromP*0.707 + 4.25)^2);
    Init3 = sqrt((LfromP*0.707 + 5.5)^2 + (LfromP*0.707 + 4.25)^2);
    x = -5.5:0.015625:5.5;
    y = -4.25:0.015625:4.25;

    for j = 1:545
        for i = 1:705
            % Calculates number of steps needed to reach each coordinate
            % from initial coordinates (0, 0) assuming each thread is
            % wrapped around a .25 inch diameter with 200 steps per rev
            LookupTable(i, j).motor1 = round((sqrt((LfromP - x(i) + 5.5)^2 + y(j)^2) -
            Init1) / 0.003927);
            LookupTable(i, j).motor2 = round((sqrt((LfromP*0.707 + x(i) + 5.5)^2 +
            (LfromP*0.707 - y(j) + 4.25)^2) - Init2) / 0.003927);
            LookupTable(i, j).motor3 = round((sqrt((LfromP*0.707 + x(i) + 5.5)^2 +
            (LfromP*0.707 + y(j) + 4.25)^2) - Init3) / 0.003927);
        end
    end
end
```

7.3 Arduino Code

```
    /* This program communicates with the SpyderCam MATLAB GUI
 * and sends information to stepper motors based on either
 * G-Code inputs or movement of the joystick.
 */
#include <AccelStepper.h>
```

```
#define joyX A0
#define joyY A2
int mode = 0;       // Denotes Joystick vs G-Code mode
int xValue, yValue, B1Value, B2Value;

AccelStepper motor1 (AccelStepper::FULL4WIRE, 2, 3, 4, 5);
AccelStepper motor2 (AccelStepper::FULL4WIRE, 6, 7, 8, 9);
AccelStepper motor3 (AccelStepper::FULL4WIRE, 10, 11, 12, 13);
void setup() {
  motor1.setMaxSpeed(1200);
  motor2.setMaxSpeed(1200);
  motor3.setMaxSpeed(1200);
  Serial.begin(9600);
}

void loop()
{
  String token1, token2, token3, token4;

  if (Serial.available() > 0)
  {
    token1 = Serial.readStringUntil('\n');  // Reads first input

    if (token1 == "G-Code")
      mode = 0;               // Switches to G-Code mode

    else if (token1 == "Joystick")
      mode = 1;               // Switches to Joystick mode

    else
    {
      token2 = Serial.readStringUntil('\n');
      token3 = Serial.readStringUntil('\n');
      token4 = Serial.readStringUntil('\n');  // Reads next four values (steps and rpm) as
                                              // strings

      int steps1 = token1.toInt();
      int steps2 = token2.toInt();
      int steps3 = token3.toInt();
      int rpm = token4.toInt();
      motor1.move(steps1);
      motor2.move(steps2);
      motor3.move(steps3);
      while(motor1.currentPosition() != motor1.targetPosition() && motor2.currentPosition()
      motor2.targetPosition() && motor3.currentPosition() != motor3.targetPosition()){
         motor1.setSpeed(rpm * 3);
```

```
        motor1.runSpeedToPosition();
        motor2.setSpeed(rpm * 3);
        motor2.runSpeedToPosition();
        motor3.setSpeed(rpm * 3);
        motor3.runSpeedToPosition();
      }
    }
  }

  if (mode == 1)
  {
    yValue = analogRead(joyY);
    if(yValue > 530)                  // If joystick is pushed up
      Serial.println("U");

    else if(yValue < 510)             // If joystick is pushed down
      Serial.println("D");


    xValue = analogRead(joyX);
    if(xValue > 510)                  // Pushed right
      Serial.println("R");

    if(xValue < 490)                  // Pushed left
      Serial.println("L");
  }
}
```