Balloon Solar Satellite (BoSS)

Founders

Sergei Bilardi, Eliza Gazda, Jessica Hagan, William Fanning

Contents

1 Systems Overview

1.1 Mission Statement

To build an Environmental Sensing Sun tracker prototype that attaches to a balloon satellite payload. With a serial connection, real time thermal and circuit data is provided in the Ground Station for the users benefit.

1.2 Requirements

The objective is to design and build a working Environmental Sensing Sun tracker prototype that is self-sufficient with its own power supply, and real time thermal and circuit data transfer through a serial connection. With operating temperatures between 60°C and -20°C, it is required to have two thermistors on either side of the structure, as well as provide this thermal information to the user. A battery source and voltage regulator must be used to control input voltage, as well as real time battery voltage and current draw must be supplied to the user in the Ground Station. The Sun Tracker must have the ability to rotate in the direction of the primary light source without electrical or structural interference, and the angle of deviation from a specified origin must be supplied to the user through the Ground Station. This angle measurement must be accurate within ten degrees. The Ground Station program must be done through MATLAB, where the thermistor, voltage and current draw, sun angle data is graphed as a function of time in seconds. An LED is used for battery life verification in the prototype and circuit protection in the forms of a zener diode circuit with a fuse backup system.

2 Electrical

2.1 Thermistor Calibration and Circuit

The experimental process of calibrating the two thermistors involved measuring the thermistor resistivity at different temperatures to determine the B constant shown in Equation 1. To do this, water is heated and brought to a maximum temperature. The two thermistors that were calibrated were placed in this water along with a temperature probe to read accurate thermal data. Using two multimeter's for each thermistor, the resistance of each thermistor as well as the temperature at which they are measured at three minute increments. After a total of 30 minutes, enough data points are taken to provide accurate data. Each measured thermistor resistance and temperature data point is used to find the constant B value, and the average of those values is taken to properly calibrate the two thermistors.

In the experiment, the initial temperature was measured at 24.5 °C, with a measured initial resistance of 9880 Ω in thermistor 1 and 9830 Ω in thermistor 2. The list of thermistor and measured resistance values are shown in Table 1 in Appendix D. From left to right, the

measurement data point times are listed up to thirty minutes in increments of three minutes. The measured temperature in Celsius is listed corresponding to the time, with a maximum temperature of 72 °C and minimum temperature of 47.7 °C. The measured temperature is then converted to Kelvin in the next column. Thermistor 1 and 2's measured resistance in KΩ are in the columns following, respectively.

To find the required B constant, Equation 2 is used. In here, the measured thermistor resistance, initial thermistor resistance, measured temperature, and initial temperature are used to find the constant. This equation is rearranged from Equation 1. In Table 2 in Appendix D, the B values are calculated based on the measured numbers in Table 1. The average B value is then taken and used for further calculations.

Based on the now known B value, and initial temperature and thermistor resistance, the required temperature values are listed with their respective calculated thermistor resistance values. This information is provided in Table 4 in Appendix D.

The calibration tables in Apppendix D are represented in plots for each thermistor, located in Appendix C. Figures 2 and 3 show the raw data collected as points and the calibration and ideal (determined from thermistor data sheet) curve as lines with their respective, determined B values.

2.2 Voltage and Current Sensors

To measure output voltage without damaging the ATmega, it was necessary to cut down the voltage. To do this, two voltage dividers before and after a 1Ω resistor were used. In the voltage dividers used two 5.1 KΩ resistors. To calculate the voltage and current, the potential difference of the two voltage dividers is calculated, as well as the current across the 1Ω resistor.

2.3 Power Indicator

To indicate successful battery life, an LED circuit is put in-between the voltage regulator and the rest of the circuit. This LED is placed on the outside of the structure to provide the user with the ability to see this information without having to open up the structure.

3 Structure

The prototype structure consists of three main sections. First, the outside body is designed in CAD and two parts are 3D printed. Second, the body is covered in a space blanket for thermal protection and contains various sensors for operation outside of the main body. Third, the inside components and circuits allow for command and data handling of the system. Each section is described in detail.

3.1 3D Printed Structure

The main structure of this prototype is a simple model similar to a 2U CubeSat design. This design is 3D printed. The 3D print of the structure allows for a lighter, less expensive body and easy modifications during construction of the prototype. The weight is especially important to increase the torque ability of the stepper motor that holds the main body and allows for tracking of the light.

The 3D print consists of two parts. The first part is a top to a box that allows opening and closing of the prototype using friction between the walls of part 1 and 2. The top also contains a slip ring, allowing the satellite to spin without tangling wires. Part 2 of the print is a 19.8mm x 9.3mm cube shown in Appendix F.1. The front of the cube contains 4 sets of 2 1.5mm holes for each lead of each sensor; these sensors are 3 photo-resistors and 2 thermistors. The front also contains shades to allow for better accuracy of tracking light. An IR sensor is placed on the front of the cube to show a warning of an approaching object. One side of the cube has 2 holes for 2 leads of a LED sensor that indicates power in the cube. The back of the cube contains opening for a second thermistors. The whole cube is screwed into a stepper motor from the bottom allowing a very stable connection between the motor and the cube. This cube has a large tolerance for shock due to screwed in parts.

3.2 Thermal Protection and Sensors

The main sensors in this design include 3 photo-resistors, 2 thermistors, 1 power indicator (LED), 1 IR sensor and a power/current sensor. The photo-resistors are on the front of the main structure of the prototype. They are lined up horizontally and separated by shades allowing the most efficient detection of light. There are two thermistors, one in the front and one in the back of the cube allowing the determination of temperature on both sides of the cube. An LED on the side of the cube allows easy detection that the power is still being delivered to the system. The power/current sensor is inside the circuit. The whole structure is covered in a blanket to allow thermal protection in high temperature environment.

3.3 Internal Setup

The internal setup of the cube contains a breadboard with all circuits, a battery pack, and all sensors, some sensors are pulled through to the outside of the cube. The breadboard and batteries are laid out on the floor of the cube and the sensors are all attached to the circuit.

4 Micrcontroller Software

The onboard microcontroller was progammed with code allowing it to use all sensors described so far and meet the requiredments for this assignment. The microcontroller software

was made to work with an ATmega644PA microcontroller. Figures 9 and 10 in Appendices H.1 and H.2 outline the functions of the code written and how it is organized, respectively.

Foe components such as stepper motors, thermistors, IR sensors and photoresistors, each header file contained a corresponding structure for these componensts which contained variables to store component parameters and settings such as which pins they were connected to other user defined parameters. For example, a stepper motor structure would be initialized for a specific stepper motor. Upon initialization, its stepping sequence, number of steps per revolution and stepping mode and pin connections, specified by the programmer, would be initialized to the structure. Additional functions for these structures are used to perform BoSS' functions. If the stepper motor is to be moved in a certain direction, a pointer to its structure would be passed to the moveMotor() function, where a direction variable, previously set, would determine which direction the motor would turn. Appendices H.3 through H.11 show the source code used.

5 Ground Station

The ground station for BoSS was coded in MATLAB, and was designed to display data from BoSS both graphically and as text. Data displayed by the ground station includes temperatures for both of our thermistors in Celsius, voltage of the power source, payload current, and sun angle. Also displayed is the elapsed time of the session, status updates from BoSS, and proximity warnings. The temperatures, voltage, and current are displayed as line graphs and as text, while the sun angle is displayed as polar plot, set up to act as a compass, and as text. All text values are displayed in the bottom right panel of the ground station as way to quickly assess the current data points being received by the ground station. All data is displayed in real time, every 0.1 seconds the ground station is updated with the latest data points. Data points are automatically saved to a .csv file.

5.1 MATLAB Code

The ground station consists of two windows, the initial window and the main window with the ground station. The initial window was created using GUIDE, a development environment used to graphically design user interfaces, rather than creating them in MATLABs code editor. The initial window asks the user for the baud rate, and the serial port that BoSS is plugged into. After clicking Connect, the initial window closes and the main window opens. The program then begins reading data from the serial port, saving the data to a .csv file, and displaying the data on the main window graphically and as text. The program will continue to display data until the window is closed, closing all serial ports and ending the program.

The plots all use MATLABs plot functions, except for the polar sun angle plot. The version of MATLAB used was R2015b, in order to comply with the computers in the Microcomputers

lab. This older version of MATLAB has an implementation of polar plots that differ from how it implements line plots, requiring some workarounds to update the data in the sun angle plot. The sun angle plot, created using the compass function, only accepts values in Cartesian coordinates, not in polar. To display the sun angle correctly, the X value and Y value of the compass plot were entered in as cosd(Sun Angle) and sind(Sun Angle), with cosd() and sind() being cosine and sine functions that accept values in degrees. Next, to display the angle as a line instead of a point, the data was set as two arrays, 0 being the first value for both arrays, and the previously mentioned X and Y coordinates as the second and final values. Next, to display the angle as a line instead of a point, the data was set as two arrays, 0 being the first value for both arrays, and the previously mentioned X and Y coordinates as the second and final values.

A problem with early versions of the code was extreme amounts of delay in updating the ground station, after a period of time (usually 60 seconds). This problem was not fixed before the demonstration on April 26, but afterwards the fault in the code was located and the problem was patched. The ground station now updates every 0.1 seconds smoothly, even after an hour of testing.

6 Testing, Troubleshooting and Resolution

The prototype was tested multiple times throughout the design, assembly and calibration of sensors and a main 1 hour test was conducted with the completed and assembled design.

6.1 Final One Hour Test and Results

The one hour test was successful, each sensor was tested with varying values, and data collected over the course is represented in the plots shown in Figure 1 in Appendix A. One main problem was the heating of the spacecraft due to electronics had an effect on the temperature probes. Over time there was an increase in the temperature of the probes as the duration of mission continued.

Appendices

A Final One Hour Test Results

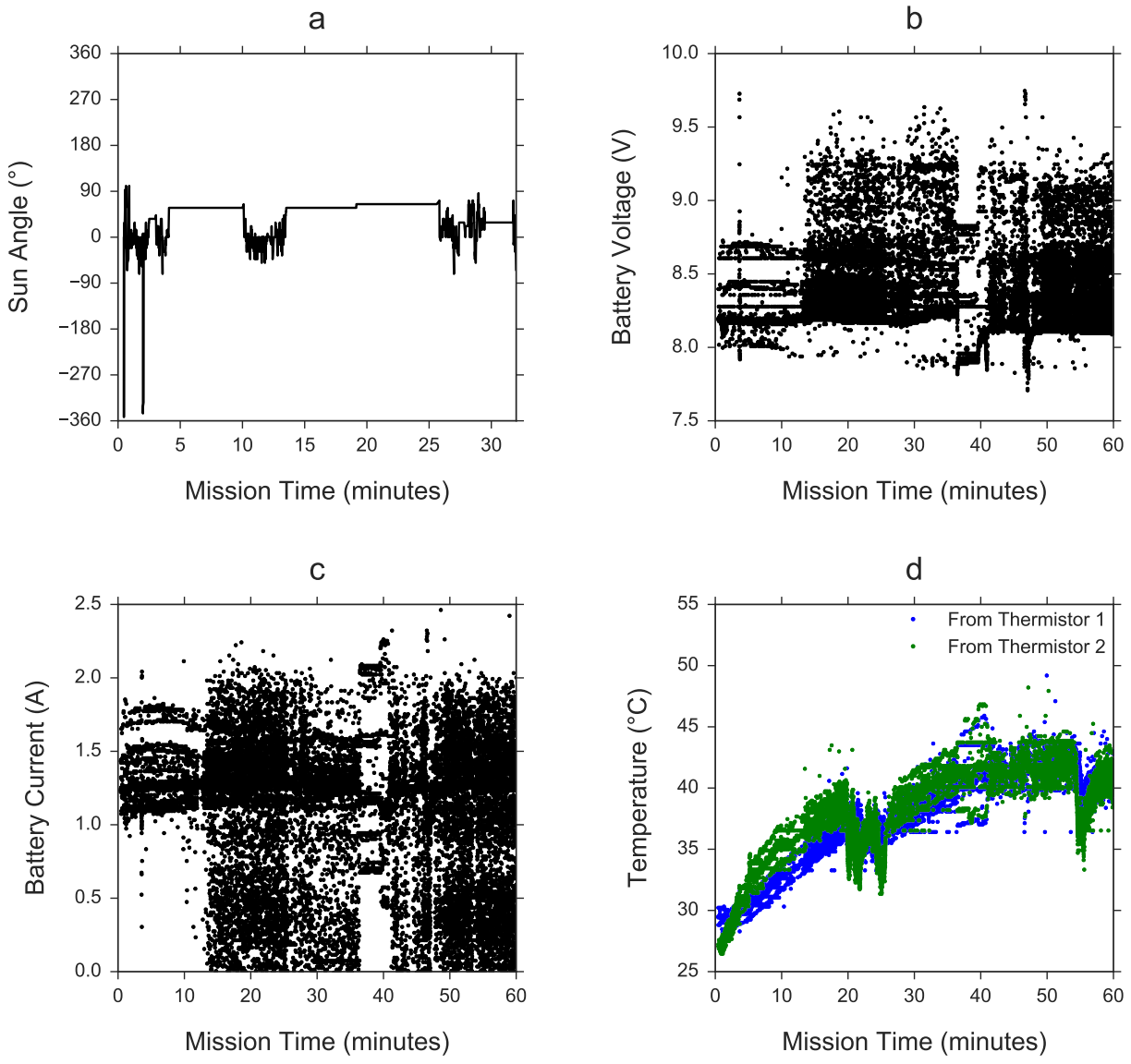

Figure 1: Shows the final results from the one hour test of the bus. Subplot a is the orientation of the bus reference off the sun angle. It can be seen where it reacquires the sun and resets the angle to zero from the sharp increases and decreases to 0°. Subplot b, c, and d are the battery voltage, battery current draw and external temperature, respectively.

B Thermal Calculations

$$R_t = R_o e^{B\left(\frac{1}{T} - \frac{1}{T_o}\right)} \tag{1}$$

$$B = \log\left(\frac{R_t}{R_o}\right) \frac{1}{\frac{1}{T} - \frac{1}{T_o}} \tag{2}$$

$$T = \left(\log\left(\frac{R_t}{R_o}\right) \frac{1}{B} - \frac{1}{T_o}\right)^{-1} \tag{3}$$

C Thermal Graphs

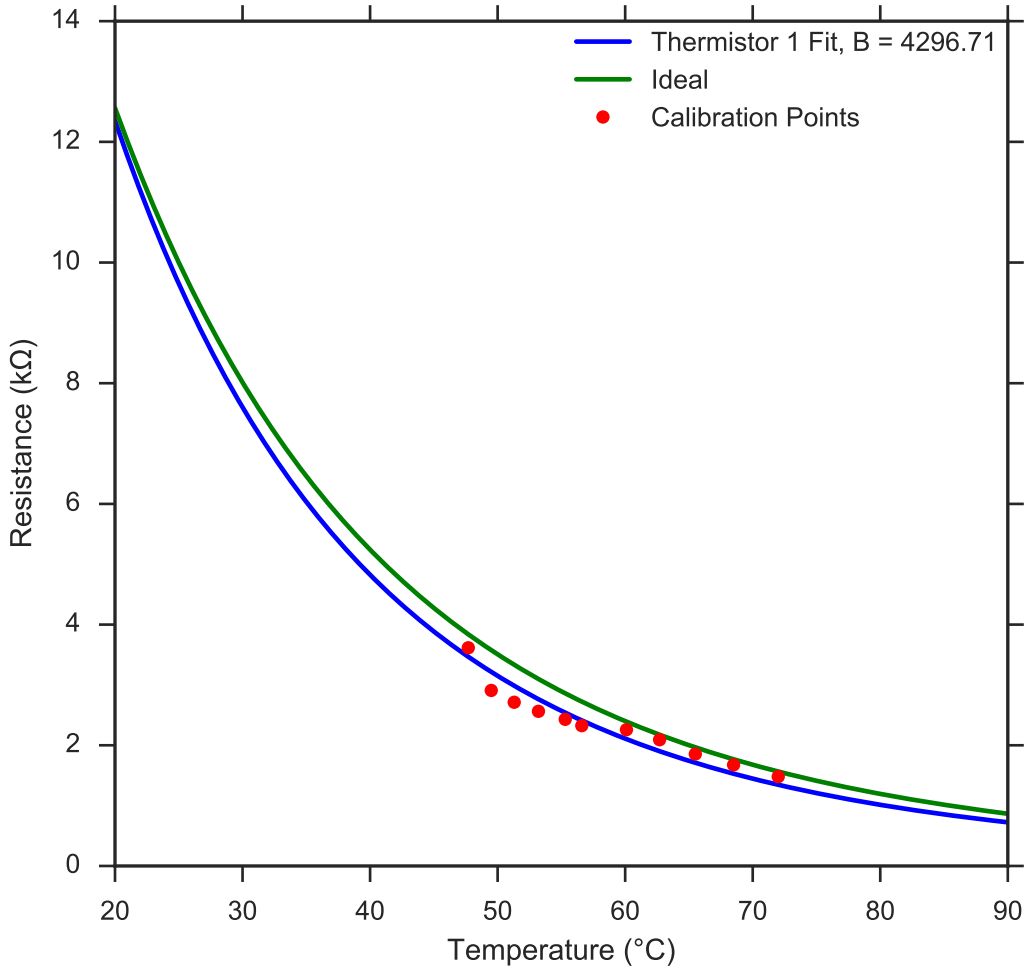

Figure 2: Shows the calibration data for thermistor 1 as points and its corresponding calibration curve using the determined B value shown above. The ideal calibration data, provided by the thermistor producer, is shown in green.

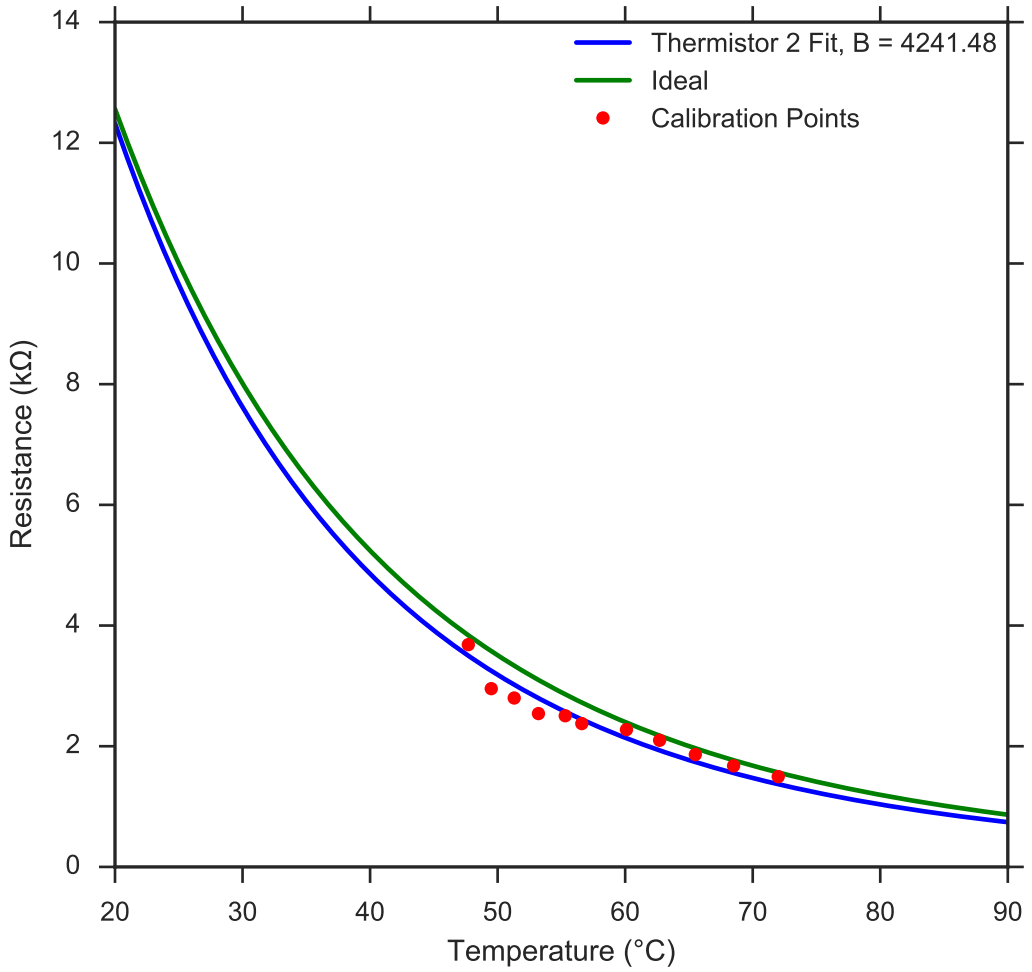

Figure 3: Shows the calibration data for thermistor 2 as points and its corresponding calibration curve using the determined B value shown above. The ideal calibration data, provided by the thermistor producer, is shown in green.

D Thermal Tables

Temperature (°C)	(K)	R_{T_1}	R_{T_2}
72.0	345.0	1.480	1.496
68.5	341.5	1.675	1.675
65.5	338.5	1.855	1.862
62.7	335.7	2.092	2.098
60.1	333.1	2.258	2.275
57.6	330.6	2.325	2.375
55.3	328.3	2.430	2.505
53.2	326.2	2.563	2.541
51.3	324.3	2.713	2.799
49.5	322.5	2.909	2.954
47.7	320.7	3.616	3.683

Table 1: Shows the measured values of temperature and resistance of both thermistor one and thermistor two at specific time data points are shown here. As temperature decreases, both thermistors show similar values of increasing resistance. The measured resistance of both thermistors are in KΩ.

B_1	B_2
4102.19	4068.00
4097.79	4086.08
4108.29	4086.58
4058.60	4037.85
4108.73	4073.73
4299.01	4220.71
4447.82	4335.34
4562.56	4574.55
4562.81	4522.20
4692.43	4614.05
4133.58	4037.21
Average	
4296.71	4241.48

Table 2: Calculated B Values with Average: These numbers are based on Table 1's measured temperature and thermistor resistance. In here, B1 is thermistor ones B value, and B2 is thermistor twos B value.

R_{T_1}	R_{T_2}	Calculated T_1	Calculated T_2
1.480	1.496	342.52	342.76
1.675	1.675	339.18	339.66
1.855	1.862	336.47	336.80
2.092	2.098	333.33	333.64
2.258	2.275	331.37	331.53
2.325	2.375	330.62	330.42
2.430	2.505	329.50	329.05
2.563	2.541	328.16	328.69
2.713	2.799	326.74	326.25
2.909	2.954	325.02	324.90
3.616	3.683	319.75	319.50

Table 3: Shows the measured thermistor resistance with respective calculated temperature. This information is used for Figures 2 and 3, and take the raw measured thermistor resistance data and the calculated average B value and produce a calculated temperature value with the respective thermistor resistance value.

Temperature (K)	Calculated R_{T_1} (kω)	Calculated R_{T_2} (kω)
340	1.62	1.66
335	1.96	2.00
330	2.38	2.43
325	2.91	2.96
320	3.58	3.63
315	4.43	4.47
310	5.52	5.56
305	6.93	6.96
300	8.76	8.77
295	11.17	11.15
290	14.35	14.29
285	18.61	18.46
280	24.37	24.08
275	32.21	31.72
270	43.01	42.21
265	58.08	56.77
260	79.33	77.23
255	109.69	106.35
250	153.64	148.32

Table 4: Known Temperature with Respective Calculated Thermistor Resistance Values: The resistance values are calculated using Equation (1) and their respective B values, and are measured in KΩ. The temperature values are measured in Kelvin and are between 60°C and -20 °C, which are the required operating temperatures.

E Electrical Circuit

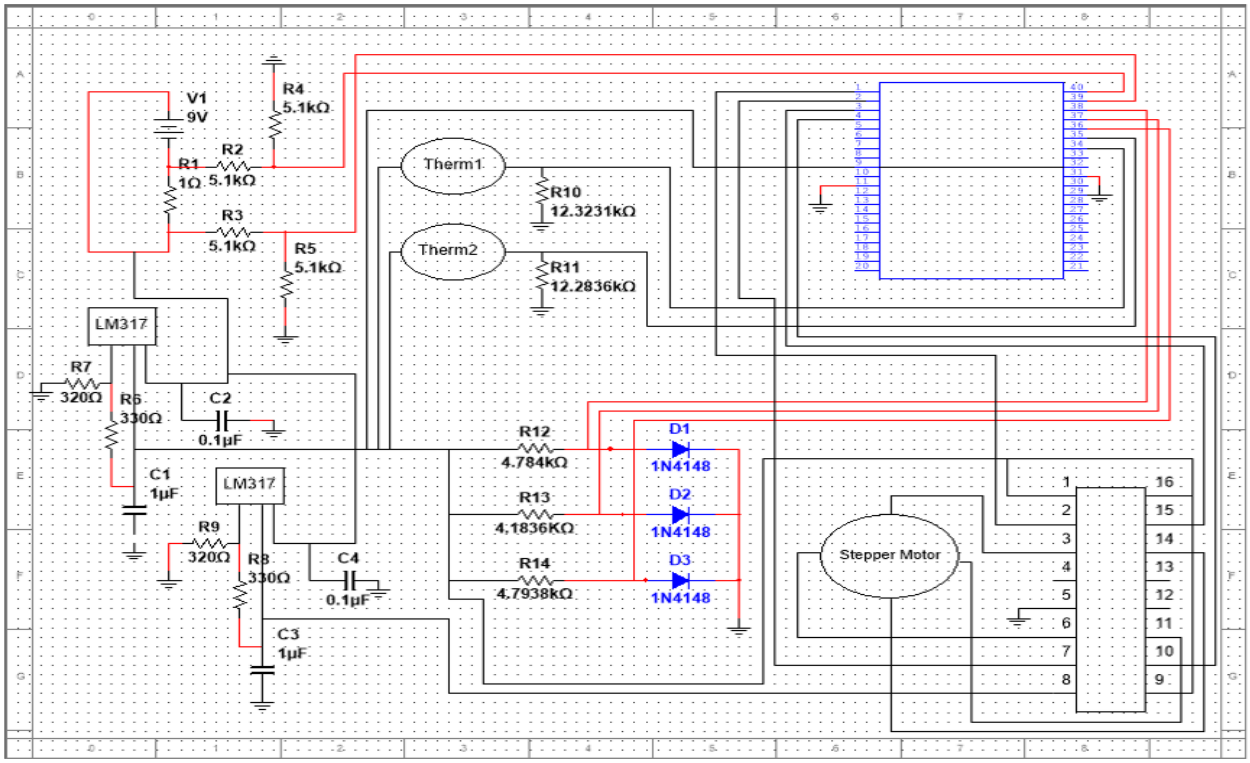

Figure 4: Shows the circuit schematic of the entire electrical system. This includes the two voltage regulators, three photoresistors, two thermistors, ATMega, H-bridge, and stepper motor.

F Structure

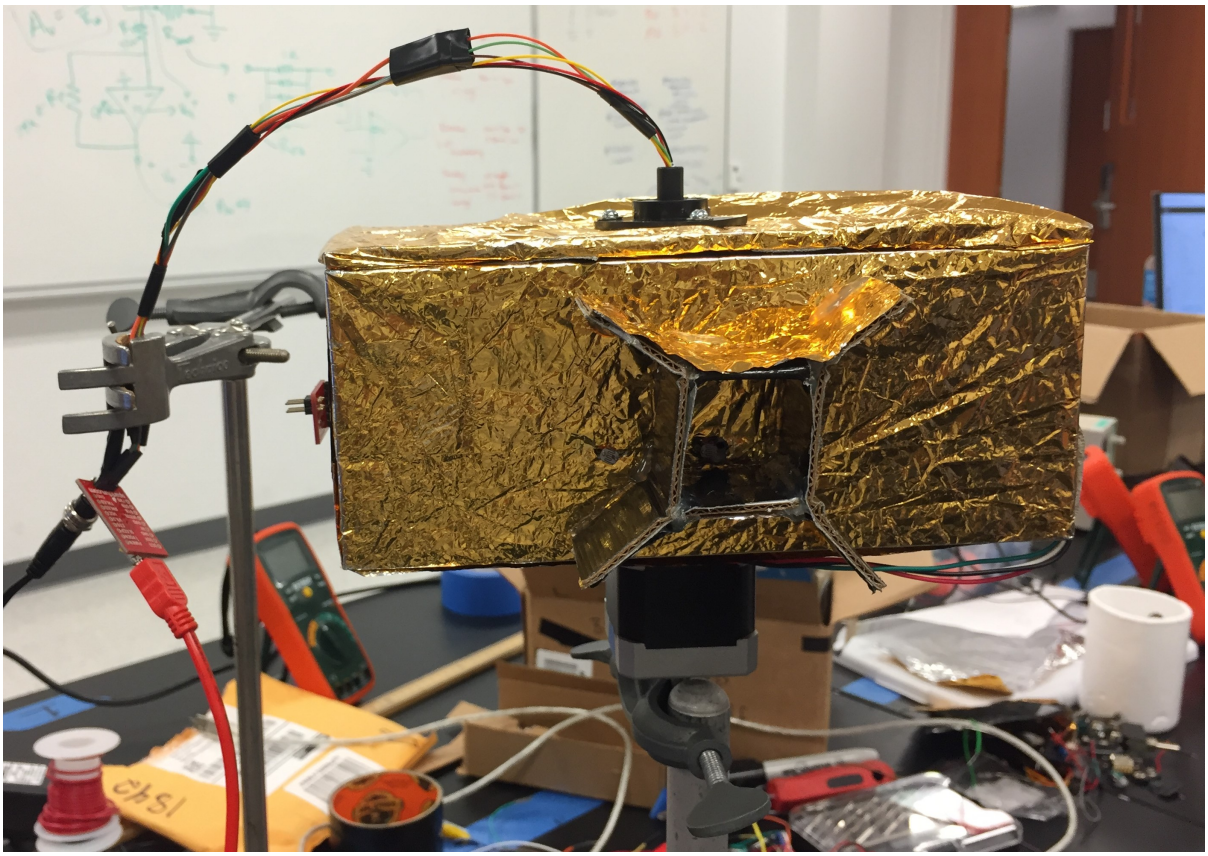

Figure 5: Shows the completed bus with a slip ring mounted on the top to prevent cord wrap and UART and external power connections held by the clamp on the left. The bus is mounted to the stepper motor through screws and is mounted to a pole for testing.

F.1 3D CAD Model

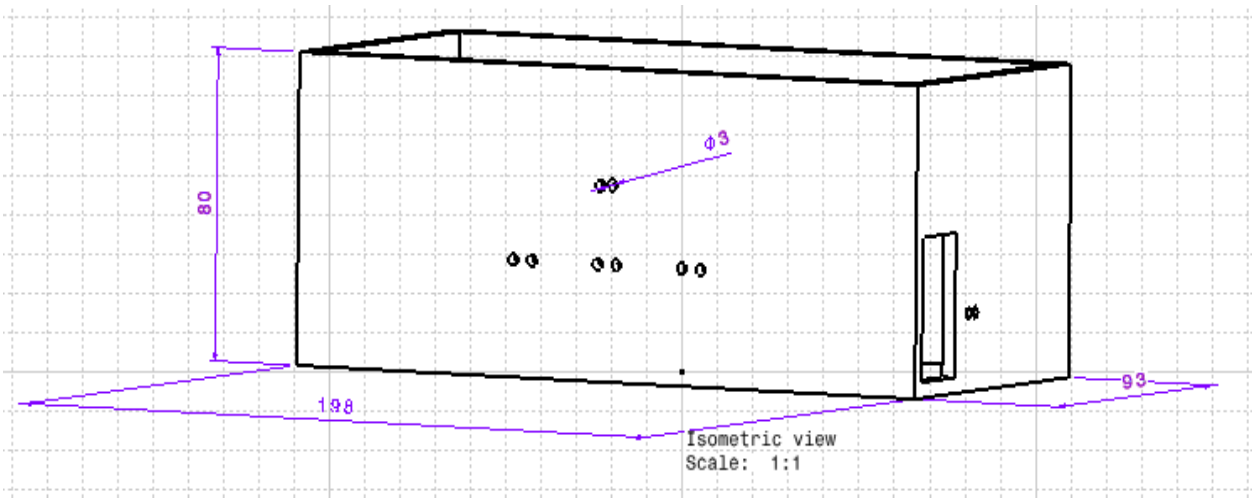

Figure 6: Shows the 3D printed enclosure which housed the electronics and batteries.

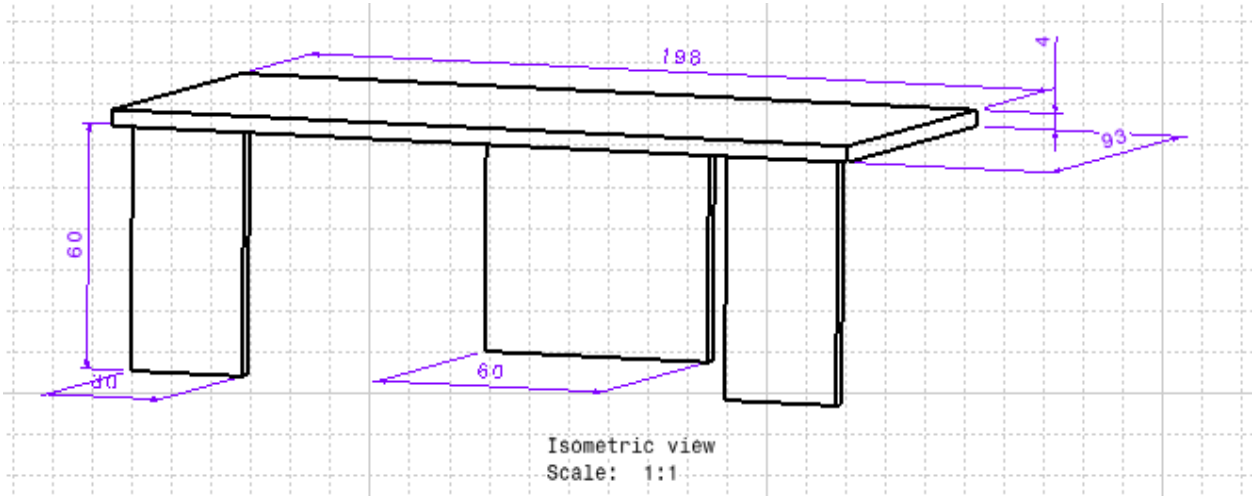

Figure 7: Shows the 3D printed lid for the enclosure shown in Figure 6.

G MATLAB GUI Source Code

The following is the source code for the MATLAB GUI developed to receive and plot the data received from the bus. This ".m" and the associated ".fig" files along with the one hour test data are available on GitHub, `https://github.com/sjbilardi/MATLABGroudStationSourceCode`.

G.1 MATLAB Code Diagram

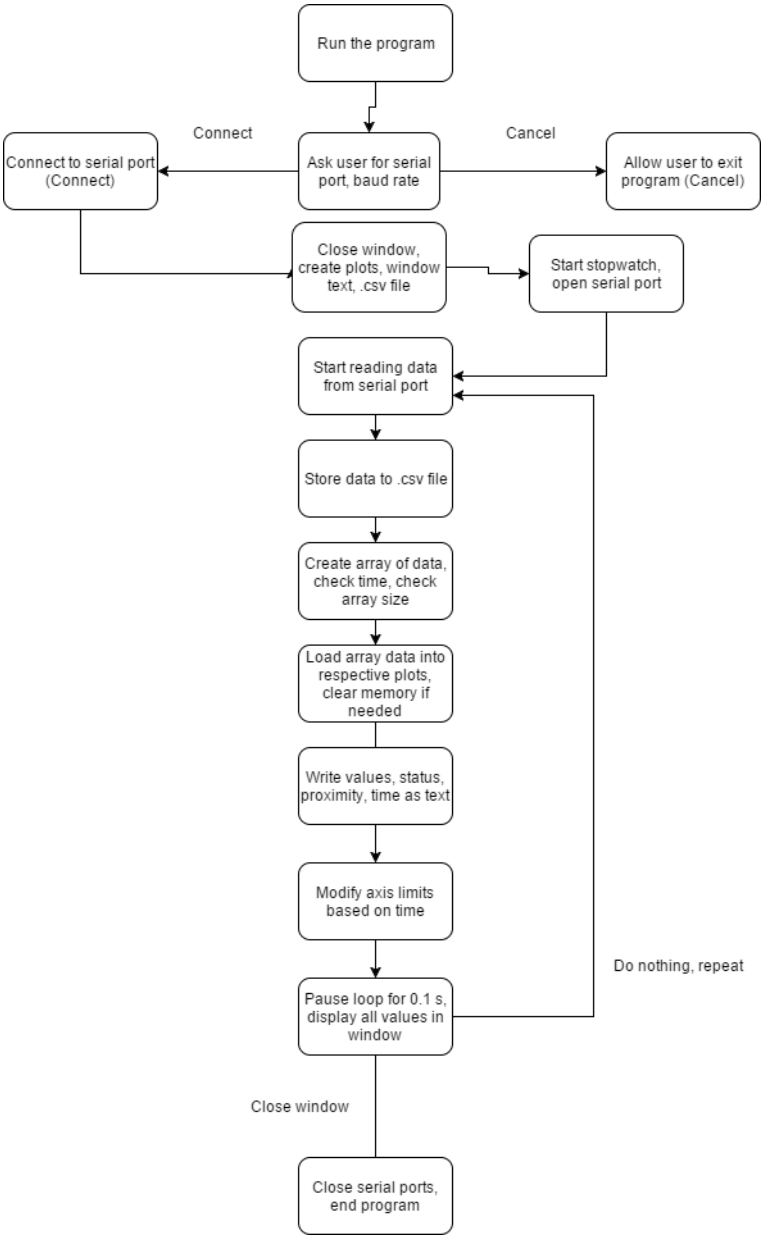

Figure 8: Shows the primary functions of the MATLAB GUI software to receive and display data from the bus.

G.2 Main MATLAB Code (mctrst.m)

```
function varargout = mctrst(varargin)
% MCTRST MATLAB code for mctrst.fig
%      MCTRST, by itself, creates a new MCTRST or raises the existing
%      singleton*.
%
%      H = MCTRST returns the handle to a new MCTRST or the handle to
%      the existing singleton*.
%
%      MCTRST('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MCTRST.M with the given input arguments.
%
%      MCTRST('Property','Value',...) creates a new MCTRST or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before mctrst_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to mctrst_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help mctrst

% Last Modified by GUIDE v2.5 10-Apr-2017 10:55:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mctrst_OpeningFcn, ...
                   'gui_OutputFcn',  @mctrst_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
```

```
end
% End initialization code - DO NOT EDIT


% --- Executes just before mctrst is made visible.
function mctrst_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to mctrst (see VARARGIN)

% Choose default command line output for mctrst
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes mctrst wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = mctrst_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in connectserial.
function connectserial_Callback(hObject, eventdata, handles)
% hObject    handle to connectserial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% clears and opens all serial ports that are currently active for the
% computer

delete(instrfindall);

% grab serial port, baud rate, from respective callback functions. set baud
% rate equal to user defined baud rate
```

```
sp = get(handles.serialport, 'String');
br = str2double(get(handles.baudrate, 'String'));
s = serial(sp);
set(s,'BaudRate',br);

% close initial window
close(gcf);

% define labels, plot titles, and delay
xLab = 'Elapsed Time (s)';
delay = 0.1;

plotTitleT1 = 'Therm 1 Temp';
yLabT1 = 'Temperature (C)';

plotTitleT2 = 'Therm 2 Temp';
yLabT2 = 'Temperature (C)';

plotTitleV = 'Battery Voltage';
yLabV = 'Voltage (V)';

plotTitleC = 'Payload Current';
yLabC = 'Current (A)';

plotTitleSA = 'Sun Angle';

% define variables
time = 0;
data = 0; % dummy variable, used to create plots
count = 0;

figure('name', 'Ground Station', 'Position', [100, 100, 1249, 695]);

% create plots in one window
% therm 1 plot
subplot(2,3,1);
ax1 = subplot(2,3,1); % put axis into variable, used later
plotTemp1 = plot(time,data);
title(plotTitleT1);
xlabel(xLab);
ylabel(yLabT1);
axis([0 10 0 1]);

% therm 2 plot
```

```
subplot(2,3,2);
ax2 = subplot(2,3,2);
plotTemp2 = plot(time,data);
title(plotTitleT2);
xlabel(xLab);
ylabel(yLabT2);
axis([0 10 0 1]);

% voltage plot
subplot(2,3,3);
ax3 = subplot(2,3,3);
plotVolt = plot(time,data);
title(plotTitleV);
xlabel(xLab);
ylabel(yLabV);
axis([0 10 0 1]);

% current plot
subplot(2,3,4);
ax4 = subplot(2,3,4);
plotCurrent = plot(time,data);
title(plotTitleC);
xlabel(xLab);
ylabel(yLabC);
axis([0 10 0 1]);

% angle plot
subplot(2,3,5);
plotAngle = compass(cosd(0),sind(0));
t = title(plotTitleSA);
set(t,'Position',[1.15 0 0]);  % move up slightly
view(90, -90);
set(findall(gcf, 'String', '  1'),'String', ' ');
set(findall(gcf, 'String', '  0.8'),'String', ' ');
set(findall(gcf, 'String', '  0.6'),'String', ' ');
set(findall(gcf, 'String', '  0.5'),'String', ' ');
set(findall(gcf, 'String', '  0.4'),'String', ' ');
set(findall(gcf, 'String', '  0.2'),'String', ' ');

% grab position of last plot, change horizontal position for text values
h = get(subplot(2,3,5), 'Position');
h(1) = h(1) + 0.2808;

% create panel, displaying text values
hp = uipanel('Position',h);
```

```
hpp = uicontrol('Parent',hp,'Style','text',...
    'FontSize',12,'Position',[20 300 60 20]);

% define labels for text values
T1st = 'Therm 1 Temp: ';
T2st = 'Therm 2 Temp: ';
Vst = 'Voltage: ';
Cst = 'Current: ';
SAst = 'Angle: ';
MT = 'Mission Time: ';

% define variables
T1v = zeros(1,1000000); % Therm 1 Temp value
T2v = zeros(1,1000000); % Therm 2 Temp value
Vv = zeros(1,1000000); % Voltage value
Cv = zeros(1,1000000); % Current value
SAv = 0; % Sun Angle value
StatusCode = 0; % Error code value

% grab current date & time
t = datetime;
t = datestr(t, 'yyyy-mm-dd HH.MM.SS');

% create header for .csv file
header = ['Elapsed time (s),' 'Therm 1 Temp (C),' 'Therm 2 Temp (C),' ...
    'Battery Voltage (V),' 'Payload Current (mA),' 'Sun Angle (degrees)'];

% .csv filename, using current date and time. saves as
% "collected_data_(YYYY-MM-DD HH.MM.SS)"
file = ['collected_data_(' t ').csv'];

% create .csv file
dlmwrite(file,[]);

% write header to .csv file
fid=fopen(file,'w');
fprintf(fid,'%s\n',header);
fclose(fid);

% start stopwatch
tic;

% open serial port, read data (first read usually produces garbage, this
% reads the garbage here so it doesn't cause the code to crash later
fopen(s);
```

```
dat = fscanf(s, '%f');

% arraySizeLimit = 100; % max number of points held by all arrays

movePt = 10; % how long in seconds until plots move in time

% create text displaying recent data in window

% Therm 1 temp value
T1value = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,'Position',[10 300 200 20]);
set(T1value, 'HorizontalAlignment', 'left')

% Therm 2 temp value
T2value = uicontrol('Parent',hp,'Style','text',...
    'FontSize',12,'Position',[10 275 200 20]);
set(T2value, 'HorizontalAlignment', 'left')

% Voltage value
Vvalue = uicontrol('Parent',hp,'Style','text',...
       'FontSize',12,'Position',[10 250 200 20]);
set(Vvalue, 'HorizontalAlignment', 'left')

% Current value
Cvalue = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,'Position',[10 225 200 20]);
set(Cvalue, 'HorizontalAlignment', 'left')

% Sun angle value
SAvalue = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,'Position',[10 200 200 20]);
set(SAvalue, 'HorizontalAlignment', 'left')

% Status of sun tracker
StatusReport = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,...
            'Position',[10 150 500 20]);
set(StatusReport, 'HorizontalAlignment', 'left')

% Proximity warning
proximityReport = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,...
            'Position',[10 125 500 20]);
set(proximityReport, 'HorizontalAlignment', 'left')
```

```
% Elapsed time of current session
missionTime = uicontrol('Parent',hp,'Style','text',...
            'FontSize',12,...
            'Position',[10 50 500 20]);
set(missionTime, 'HorizontalAlignment', 'left')

% while the plots window is open...
while ishandle(plotTemp1)

    % read data from serial port.
    %dat = fscanf(s, '%f');

    dataString = fscanf(s, '%s');
    dataString2 = strsplit(dataString, ',');
    data = zeros(length(dataString2),1);
    for counter=1:length(data)
        data(counter) = str2num(dataString2{counter});
    end

    if(~isempty(data) && isfloat(data))

        % increase counter by one, grab current stopwatch time
        count = count + 1;
        time(count) = toc;

        % write serial data, time, to .csv file
        dlmwrite(file,[time(count) data(1:5).'],'-append');

        % create array of serial data values
        T1v(count) = double(data(1));
        T2v(count) = double(data(2));
        Vv(count) = double(data(3));
        Cv(count) = double(data(4));
        SAv = double(data(5));
        StatusCode = int8(data(6));
        proximityCode = int8(data(7));

        % set x-axis and y-axis data for each plot
        set(plotTemp1,'XData',time(1:count),'YData',T1v(1:count),'Marker','.',...
            'Color', 'red', 'MarkerEdgeColor','red',...
            'MarkerFaceColor', 'red');
        set(plotTemp2,'XData',time(1:count),'YData',T2v(1:count), 'Marker','.',...
            'Color', 'red', 'MarkerEdgeColor','red',...
            'MarkerFaceColor', 'red');
        set(plotVolt,'XData',time(1:count),'YData',Vv(1:count),'Marker','.',...
```

```
    'Color', 'blue', 'MarkerEdgeColor','blue',...
    'MarkerFaceColor', 'blue');
set(plotCurrent,'XData',time(1:count),'YData',Cv(1:count),'Marker','.',...
    'Color', 'green', 'MarkerEdgeColor','green',...
    'MarkerFaceColor', 'green');

% set angle for sun angle plot. set() commmands
set(plotAngle, 'XData', [0 cosd(SAv(end))], 'YData', ...
        [0 sind(SAv(end))], 'Marker', '.');

% turn serial data values into strings
T1vs = num2str(T1v(count), 3);
T2vs = num2str(T2v(count), 3);
Vvs = num2str(Vv(count), 3);
Cvs = num2str(Cv(count), 3);
SAvs = num2str(SAv, 3);

% write text values
T1data = [T1st T1vs ' C'];
T2data = [T2st T2vs ' C'];
Vdata = [Vst Vvs ' V'];
Cdata = [Cst Cvs ' A'];
SAdata = [SAst SAvs ' '];
MTdata = [MT num2str(time(count)) ' s'];

% update text values in window
set(T1value,'String',T1data);
set(T2value,'String',T2data);
set(Vvalue,'String',Vdata);
set(Cvalue,'String',Cdata);
set(SAvalue,'String',SAdata);

% Status report updates, depends on number recieved.
% 0 - Sun detected to right. Moving clockwise.
% 1 - Sun detected on left. Moving counterclockwise.
% 2 - Satellite oriented towards sun. Idling orientation.
% 3 - Battery low.
% 4 - Searching for Sun.
if StatusCode == 0
    set(StatusReport,'String', ...
        'Sun detected to right. Moving clockwise.')
elseif StatusCode == 1
    set(StatusReport,'String', ...
        'Sun detected on left. Moving counterclockwise.')
elseif StatusCode == 2
```

```
            set(StatusReport,'String', ...
                'Satellite oriented towards sun. Idling orientation.')
        elseif StatusCode == 3
            set(StatusReport,'String', 'Battery low.')
        elseif StatusCode == 4
            set(StatusReport,'String', 'Searching for Sun.')
        end

        % Proximity warning updates, depends on number recieved.
        % 5 - WARNING: Object detected within 1 meter distance.
        % 6 - CLEAR: No obstacles detected.
        % other - INVALID PROXIMITY CODE.
        if proximityCode == 5
            set(proximityReport,'String', ...
                'WARNING: Object detected within 1 meter distance.')
        elseif proximityCode == 6
            set(proximityReport,'String', 'CLEAR: No obstacles detected.')
        else
            set(proximityReport,'String', 'INVALID PROXIMITY CODE.')
        end

        % update elapsed time in window
        set(missionTime, 'String', MTdata)

        % if the elapsed time is less than five seconds, axis is static, if
        % greater than five seconds, axis moves with data
        if time(count) < movePt
        axis([ax1], [0 movePt 0 100]);
        axis([ax2], [0 movePt 0 100]);
        axis([ax3], [0 movePt 0 10]);
        axis([ax4], [0 movePt 0 2]);
        else
            axis([ax1],[time(count)-movePt time(count) 0 100]);
            axis([ax2], [time(count)-movePt time(count) 0 100]);
            axis([ax3], [time(count)-movePt time(count) 0 10]);
            axis([ax4], [time(count)-movePt time(count) 0 2]);
        end

        % pause loop for delay amount, then continue
    end
    pause(delay);

    % closes serial ports when window with plots is closed
    if ~ishandle(plotTemp1)
          delete(instrfindall);
```

```
    end
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% close window when cancel button is pushed
close(gcf);


function serialport_Callback(hObject, eventdata, handles)
% hObject    handle to serialport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get serial port from user
serialp = get(hObject, 'String');
handles.serialport = serialp;
display(serialp);

% Hints: get(hObject,'String') returns contents of serialport as text
%        str2double(get(hObject,'String')) returns contents of serialport as a double


% --- Executes during object creation, after setting all properties.
function serialport_CreateFcn(hObject, eventdata, handles)
% hObject    handle to serialport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function baudrate_Callback(hObject, eventdata, handles)
% hObject    handle to baudrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% get baud rate from user
baudr = str2double(get(hObject, 'String'));
handles.baudrate = baudr;
display(baudr);

% Hints: get(hObject,'String') returns contents of baudrate as text
%        str2double(get(hObject,'String')) returns contents of baudrate as a double

% --- Executes during object creation, after setting all properties.

function baudrate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to baudrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

H Microcontroller Source Code

The following are each of the code files used to program BoSS. Note that these files are also available on GitHub, `https://github.com/sjbilardi/BalloonSolarSatelliteCodeFiles`.

H.1 Code Diagram

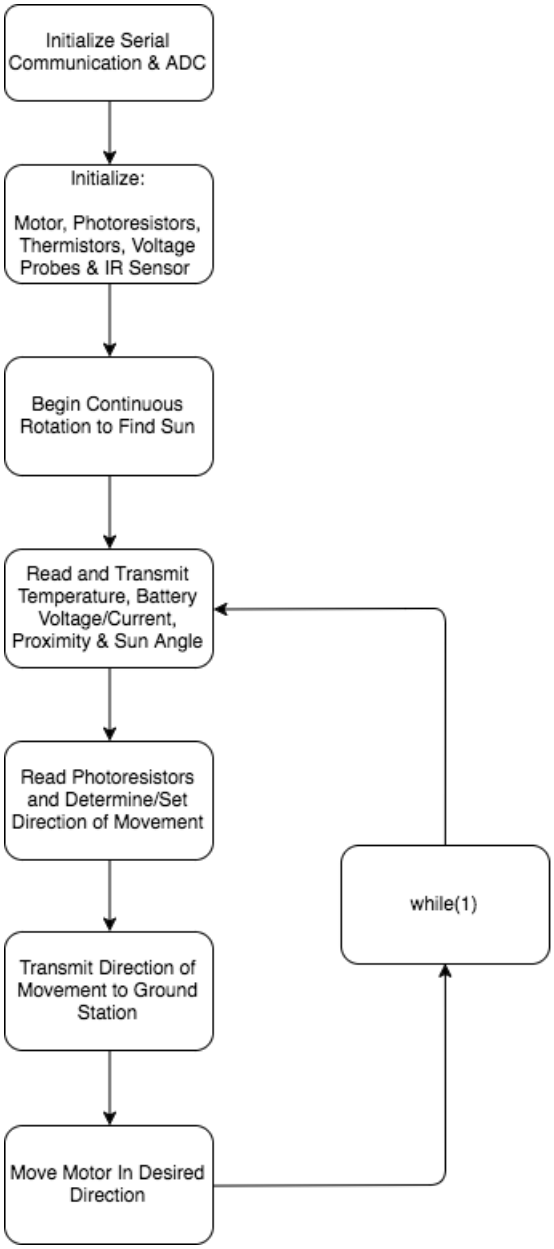

Figure 9: Shows the microcontroller code block diagram outlining the primary functions of the code.

H.2 Code File Structure

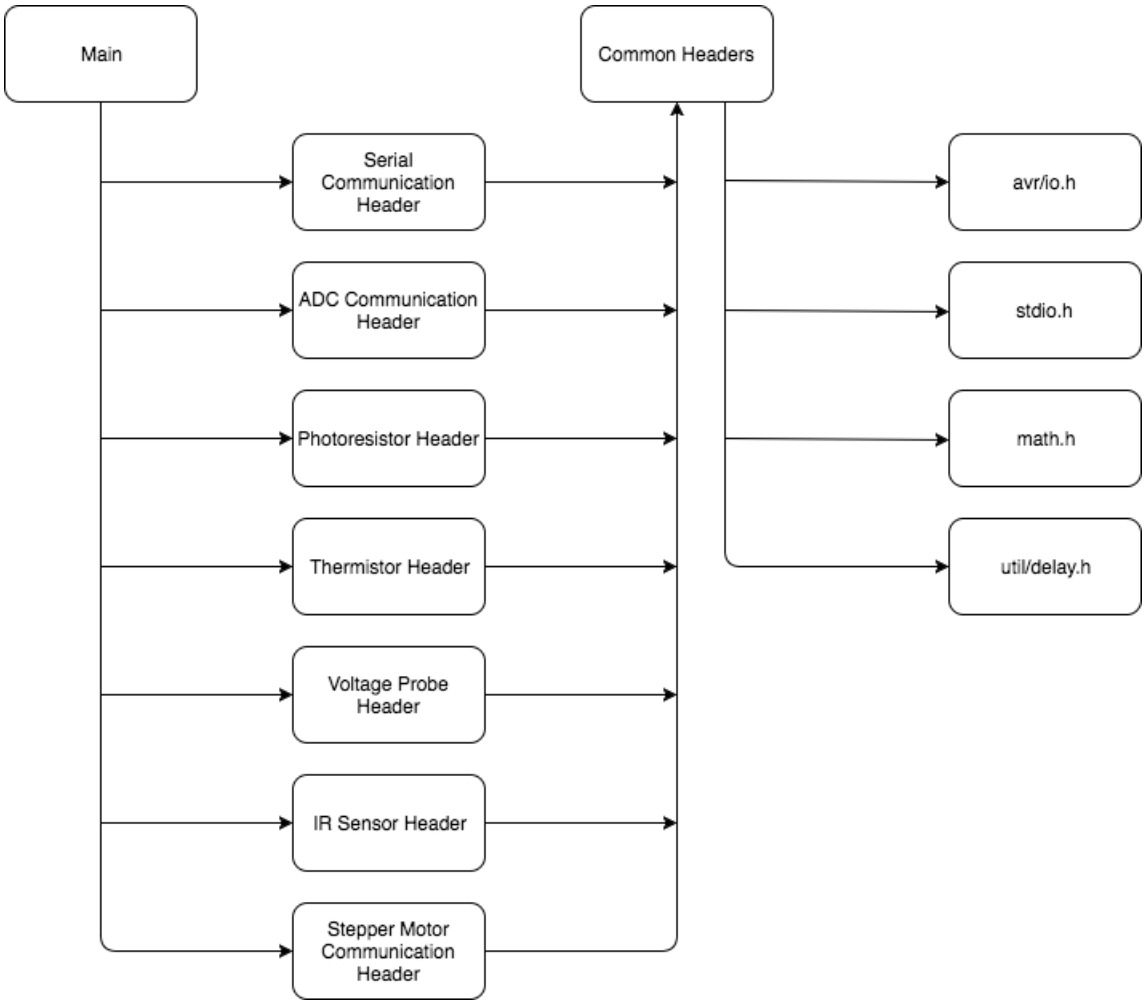

Figure 10: Shows the custom header files for each major component in the project along with the common headers used in microcontroller code.

H.3 Main Function (main.c)

```
#include "commonHeaders.h"
#include "serialComm.h"
#include "adcComm.h"
#include "LEDComm.h"
#include "thermistors.h"
#include "voltageProbes.h"
#include "distanceSensor.h"
#include "stepperMotor.h"

/* Pin Layout */
#define VPROBEBATTPIN        0 // Voltage Probe Battery
#define VPROBERESPIN         1 // Voltage Probe Resistor
#define LEDLPIN              2 // Photoresistor Left
#define LEDMPIN              3 // Photoresistor Middle
#define LEDRPIN              4 // Photoresistor Right
#define THERM1PIN        5 // Thermistor 1
#define THERM2PIN        6 // Thermistor 2
#define DISTSENSORPIN        7 // IR Distance Sensor
#define RESETSUNANGLE        1
#define KEEPSUNANGLE         0

void calibrateLEDSMotor(Motor *motor, LEDS *leds)
{
        int i;
        int j;
        char *steps;
        char stepSize;
        int numbSteps;

        /* Reset step number in motor */
        motor->steps = 0;

        if((motor->mode == FULLSTEPMODE) || (motor->mode == WAVESTEPMODE))
        {
                steps = &motor->fullSteps;
                numbSteps = motor->fullStepNumb;
                stepSize = motor->fullStepsSize;
        }
        else // if motor->mode == HALFSTEPMODE
        {
                steps = &motor->halfSteps;
                numbSteps = motor->halfStepNumb;
```

```
                stepSize = motor->halfStepsSize;
        }

        /* Set motor direction */
        motor->direction = GOCLOCKWISE;
        motor->movementAmount = BYSTEP;

        /* Record offset values */
        while(motor->steps < numbSteps)
        {
                for(i=0; i<4; i++)
                {
                        // read leds
                        for(j=0; j<sizeof(leds->pins)/sizeof(char); j++)
                        {
                                leds->offset[motor->steps][j]
                                        = readAnalog(leds->pins[j]);
                        }
                        motor->stepSeqIndex = i;
                        moveMotor(motor);
                        motor->steps = motor->steps + 1;
                }
        }

        motor->movementAmount = BYCYCLE;
        leds->direction = IDLE;
        motor->direction = IDLE;
}

void findSun(Motor *motor, LEDS *leds)
{
        char reorientationRequired = 0;
        int i;
        int numbSteps;

        if((motor->mode == FULLSTEPMODE) || (motor->mode == WAVESTEPMODE))
        {
                numbSteps = motor->fullStepNumb;
        }
        else // if motor->mode == HALFSTEPMODE
        {
                numbSteps = motor->halfStepNumb;
        }

        /*
```

```
                Keep Moving Motor Clockwise if the Middle
                Photoresistor Does Not Read SUNDETECTMULT Times the Max Deviation
        */
        if((leds->adcVal[leds->leftLED] <
                    SUNDETECTMULT*leds->offset[motor->steps][leds->leftLED]) &&
                (leds->adcVal[leds->middleLED] <
                         SUNDETECTMULT*leds->offset[motor->steps][leds->middleLED]) &&
                (leds->adcVal[leds->rightLED] <
                         SUNDETECTMULT*leds->offset[motor->steps][leds->rightLED]))
                {
                        do
                        {
                                reorientationRequired = 1;
                                motor->direction = GOCLOCKWISE;
                                getLEDSVal(leds);
                                moveMotor(motor);
                        } while(leds->adcVal[leds->middleLED] <
                                SUNDETECTMULT *
                                leds->offset[motor->steps][leds->middleLED]);
                }

        if(reorientationRequired)
        {
                motor->direction = IDLE;
                leds->direction = IDLE;
                motor->sunAngle = 0;
        }
}

int main()
{
        int i;
        char buffer[60] = {"\0"};

        // initialize the UART for serial communication with computer
        uart_init();

        /* Motor Setup */
        Motor motor = motor_init1(0x0F); // initialize motor to pins 0-3 on PORTB
        motor.mode = FULLSTEPMODE; // set motor to move using full step pattern
        motor.movementAmount = BYCYCLE;

        /* ADC Setup */
        initADC();
```

```
/* LED Setup */
LEDS leds = leds_init(LEDLPIN, LEDMPIN, LEDRPIN);

/* Thermistor Setup */
Thermistor therm1 = therm1_init(THERM1PIN);
Thermistor therm2 = therm2_init(THERM2PIN);

/* Voltage Probes Setup */
VoltageProbes voltageProbes = voltageProbes_init(VPROBEBATTPIN, VPROBERESPIN);

/* Distance Sensor Setup */
DistanceSensor distanceSensor = distanceSensor_init(DISTSENSORPIN);

/* Calibrate LEDS and Motor */
calibrateLEDSMotor(&motor, &leds);

_delay_ms(500);

/* Search for Sun */
findSun(&motor, &leds);

_delay_ms(250);

while(1)
{
        sprintf(buffer, ""); // reset buffer

        /* Read Temperature Values */
        getTemp(&therm1); // get temp1
        getTemp(&therm2); // get temp2

        /* Print Temperature Values */
        // convert double to string
        dtostrf(therm1.temp, 4, 3, buffer + strlen(buffer));
        sprintf(buffer + strlen(buffer), ",");
        dtostrf(therm2.temp, 5, 3, buffer + strlen(buffer));
        sprintf(buffer + strlen(buffer), ",");

        /* Read Voltage Values */
        readVoltageCurrent(&voltageProbes);

        /* Print Battery Voltage and Current */
        dtostrf(voltageProbes.voltage[voltageProbes.battVProbe],
                6, 3, buffer + strlen(buffer));        // battery voltage (V)
        sprintf(buffer + strlen(buffer), ",");
```

```
dtostrf(voltageProbes.current, 5, 3, buffer + strlen(buffer));
sprintf(buffer + strlen(buffer), ",");

/* Print Sun Angle */
dtostrf(motor.sunAngle, 2, 1, buffer + strlen(buffer)); // degrees
sprintf(buffer + strlen(buffer), ",");

/* Read Distance */
getDistance(&distanceSensor);

/* Read LEDs */
getLEDSVal(&leds);
for(i=0;i<sizeof(leds.adcVal)/sizeof(uint16_t);i++)
{
        if(PRINTPRVALS)
        {
                sprintf(buffer + strlen(buffer),
                        "%d ", leds.adcVal[i]);
        }
}

/* Check if Sun is Within Tracking Range and Search if it Is Lost */
findSun(&motor, &leds);

/* Move counter clockwise if left LED reads brighter light */
if(leds.direction == GOCOUNTERCLOCKWISE)
{
        // move counter clockwise
        motor.direction = GOCOUNTERCLOCKWISE;
        sprintf(buffer + strlen(buffer), "%d,", MOVINGCOUNTERCLOCK);
}

/* Move clockwise if right LED reads brighter light */
else if(leds.direction == GOCLOCKWISE)
{
        motor.direction = GOCLOCKWISE;
        sprintf(buffer + strlen(buffer), "%d,", MOVINGCLOCK);
}

/* Do nothing and idle position */
else
{
        motor.direction = IDLE;
        sprintf(buffer + strlen(buffer), "%d,", IDLING);
}
```

```
                if(distanceSensor.proximityWarning)
                {
                        sprintf(buffer + strlen(buffer), "%d", PROXIMITYWARNING);
                }
                else
                {
                        sprintf(buffer + strlen(buffer), "%d", NOOBJDETECTED);
                }

                /* Execute motor movement */
                moveMotor(&motor);

                /* Send data over serial */
                sprintf(buffer + strlen(buffer), "\n\r");
                write_uart(buffer);
        }

}
```

H.4 Common Headers (commonHeaders.h)

```
#include <avr/io.h>
#include <stdio.h>
#include <math.h>
#define F_CPU 1000000UL
#include <util/delay.h>

#define VCC 5.1 // V

/* Options for satellite direction movement */
#define GOCOUNTERCLOCKWISE  0
#define IDLE                1
#define GOCLOCKWISE         2

/* Motor step options */
#define WAVESTEPMODE        0 // waveStepMode
#define FULLSTEPMODE        1 // fullStepMode
#define HALFSTEPMODE        2 // halfStepMode
#define BYCYCLE             3 // Move one cycle when during function call
#define BYSTEP              4 // Move one step at a time
#define BYDEGREES           5 // Move number of degrees during function call

#define MOTORDELAY          30 // ms

#define SUNDETECTMULT       1.3// sun detection multiplier for max deviation

/* Comment codes */
#define MOVINGCLOCK         0 // sun detected on right led; moving clockwise
#define MOVINGCOUNTERCLOCK  1 // sun detected on left  led; moving counterclockwise
#define IDLING              2 // satellite oriented towards sun; idling position
#define BATTERYLOW          3 // battery voltage low; reaching specified limit
#define SEEKINGSUN          4 // searching for sun
#define PROXIMITYWARNING    5 // object detected within ~1 meter
#define NOOBJDETECTED       6 // no object detected; path ahead of spacecraft clear

/* Enable serial outputs */
#define DEBUG               0
#define PRINTPRVALS         0
```

H.5 Serial Communication Header (serialComm.h)

```
#include "commonHeaders.h"

#define UART_BAUD 4800
//Define Baud Rate as described in AVR manual
#define UART_UBRR_VALUE (((F_CPU / (UART_BAUD * 16UL))) - 1)

void uart_init() // initialize UART
{
        /*Set Baud rate*/
        UBRR0H = UART_UBRR_VALUE >> 8;  // set high byte
        UBRR0L = UART_UBRR_VALUE;  // set low bytes
        /*Frame format: 8 data bits, no parity, 1 stop bit*/
        UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
        /*Enable Transmit and Receive*/
        UCSR0B = (1<< RXEN0)|(1<<TXEN0);
}

void write_uart(unsigned char c[]) // transmit data over USART
{
        do
        {
                while((UCSR0A&(1<<UDRE0)) == 0); // Wait for Transmit Buffer to Empty
                UDR0 = *c; // Transmit the character
                c++; // Increment the pointer to point to the next character
        }while(*c != '\0');
}
```

H.6 ADC Communication Header (adcComm.h)

```
#include "commonHeaders.h"

void initADC()
{
        // initializes ADC for 10-bit reading

        // Set ADC prescalar to 8: Which is 125KHz sample rate @ 1MHz
        ADCSRA |= (1<<ADPS1) | (1<<ADPS0);

        ADMUX |= (1<<REFS0); // This sets ADC reference to AVCC

        ADCSRA |= (1<<ADEN); // Enable ADC
}

uint16_t readAnalog(char channel)
{
        uint16_t adcVal;

        ADMUX = (1<<REFS0) | (channel & 0x0f);  //select input and ref

        ADCSRA |= (1 << ADSC); // Start ADC Conversions

        // wait for conversion to complete
        while ((1 << ADSC) & ADCSRA)
        {
                // wait until end of conversion
        }

        adcVal = ADCL | (ADCH<<8);

        return adcVal; // return converted analog value
}
```

H.7 Photoresistor Communication Header (LEDComm.h)

```
#include "commonHeaders.h"

/* LED */
typedef struct {
        uint16_t adcVal[3];

        char leftLED;
        char middleLED;
        char rightLED;

        char direction;

        char pins[3];

        uint16_t offset[400][3]; // stores noise level for system
        } LEDS;

/* LEDS Functions */
LEDS leds_init(char leftLEDpin, char middleLEDpin, char rightLEDpin)
{
        int i;
        LEDS leds = {
                {0, 0, 0},  // default val to zero
                0, // left LED index
                1, // middle LED index
                2, // right LED index
                IDLE,
                {leftLEDpin, middleLEDpin, rightLEDpin},
                {0}};

        return leds;
}

void getLEDSVal(LEDS *leds)
{
        char i;

        /* Read Photoresistor ADC Values and Correct for Offset */
        for(i=0; i<sizeof(leds->adcVal)/sizeof(uint16_t);i++)
        {
                leds->adcVal[i] = readAnalog(leds->pins[i]);
        }
```

```
        /* Move counter clockwise if left photoresitor reads brighter light */
        if((leds->adcVal[leds->leftLED] > leds->adcVal[leds->middleLED]) &&
        (leds->adcVal[leds->leftLED] > leds->adcVal[leds->rightLED]))
        {
                leds->direction = GOCOUNTERCLOCKWISE;
        }

        /* Move clockwise if right photoresitor reads brighter light */
        else if((leds->adcVal[leds->rightLED] > leds->adcVal[leds->middleLED]) &&
        (leds->adcVal[leds->rightLED] > leds->adcVal[leds->leftLED]))
        {
                leds->direction = GOCLOCKWISE;
        }

        /* Do nothing and idle position */
        else
        {
                leds->direction = IDLE;
        }
}
```

H.8 Thermistor Communication Header (thermistors.h)

```
#include "commonHeaders.h"

/* Thermistors */
typedef struct {
        int thermNumb;

        double Ro;
        double To;
        double B;

        double Vrange;
        double R;
        double Rt;

        uint16_t adcVal;
        double Vo;
        double temp;

        char pin;
} Thermistor;

/* Motor Functions */
Thermistor therm1_init(char pin)
{
        Thermistor therm1 = {
                1,
                9.88,         // k-ohm
                297.5,         // K
                4296.71,
                VCC,
                12.3231,// k-ohm
                0,
                0,
                0.0,
                0.0,
                pin};

        return therm1;
}

Thermistor therm2_init(char pin)
{
```

```
        Thermistor therm2 = {
                2,
                9.83,         // k-ohm
                297.5,         // K
                4241.48,
                VCC,
                12.2836,// k-ohm
                0,
                0,
                0.0,
                0.0,
                pin};

        return therm2;
}

void getTemp(Thermistor *therm)
{
        /* Converts the ADC value measured from the port with a thermistor connected
           and converts the value to degrees Celsius. */

        char buffer[60];
        sprintf(buffer, "");

        // read ADC value
        therm->adcVal = readAnalog(therm->pin);

        therm->Vo = ((double)therm->adcVal)/pow(2,10) * therm->Vrange;

        therm->Rt = therm->R*(VCC/therm->Vo - 1);

        therm->temp = pow((log(therm->Rt/therm->Ro)/therm->B + 1/therm->To), -1)
                        - 273.15;

        if(DEBUG == 1)
        {
                sprintf(buffer + strlen(buffer), "ADC: %d ", therm->adcVal);

                sprintf(buffer + strlen(buffer), "V: ");
                dtostrf(therm->Vo, 4, 3, buffer + strlen(buffer));

                sprintf(buffer + strlen(buffer), " Rt: ");
                dtostrf(therm->Rt, 5, 3, buffer + strlen(buffer));

                sprintf(buffer + strlen(buffer), " T: ");
```

```
                dtostrf(therm->temp, 5, 3, buffer + strlen(buffer));

                sprintf(buffer + strlen(buffer), "\n\r");
                write_uart(buffer);
        }
}
```

H.9 Voltage Probe Communication Header (voltagesProbes.h)

```
#include "commonHeaders.h"

typedef struct {
        uint16_t adcVal[2];
        double voltage[2];
        double current;
        double R;
        double RL;
        double currentResistor;
        char battVProbe;
        char ResVProbe;
        char pin[2];
} VoltageProbes;

VoltageProbes voltageProbes_init(char pin1, char pin2)
{
        VoltageProbes voltageProbes = {
                {0, 0},
                {0.0, 0.0},        // V
                0.0,                // mA
                5.1,                // k-ohms
                5.1,                // k-ohms
                1,                 // ohms
                0,                 // battery V index
                1,                 // resistor V index
                {pin1, pin2}};
        return voltageProbes;
}

void readVoltageCurrent(VoltageProbes *voltageProbes)
{
        char i;

        /* Read ADC values from probes and convert to voltages */
        for(i=0; i<sizeof(voltageProbes->adcVal)/sizeof(uint16_t); i++)
        {
                voltageProbes->adcVal[i] = readAnalog(voltageProbes->pin[i]);
                voltageProbes->voltage[i] = (((double)voltageProbes->adcVal[i]/1024)) *
                                        VCC * (voltageProbes->R +
                                        voltageProbes->RL)/voltageProbes->RL; // V
        }
```

```
        /* Calculate the current using the voltage values */
        voltageProbes->current = (fabs(voltageProbes->voltage[0] -
                                       voltageProbes->voltage[1]) /
                                       voltageProbes->currentResistor); // mA
}
```

H.10 Distance IR Sensor Communication Header (distanceSensor.h)

```
#include "commonHeaders.h"

typedef struct {
        uint16_t adcVal;
        double distance;
        char proximityWarning;
        char pin;
} DistanceSensor;

DistanceSensor distanceSensor_init(char pin)
{
        DistanceSensor distanceSensor = {
                0,
                0.0,        // cm
                0,
                pin};

        return distanceSensor;
}

void getDistance(DistanceSensor *distanceSensor)
{
        distanceSensor->adcVal = readAnalog(distanceSensor->pin);

        distanceSensor->distance = 9801.8*pow(((double)distanceSensor->adcVal),
                -1.127); // use power function to calibrate ADC to distance (cm)

        if(distanceSensor->distance < 90.0)
        {
                distanceSensor->proximityWarning = 1;
        }

        else
        {
                distanceSensor->proximityWarning = 0;
        }
}
```

H.11 Stepper Motor Communication Header (stepperMotor.h)

```
#include "commonHeaders.h"

/* Motor */
typedef struct {
        int waveSteps[4];
        int waveStepNumb;
        double waveStepRes;

        int fullSteps[4];
        int fullStepNumb;
        double fullStepRes;

        int halfSteps[8];
        int halfStepNumb;
        double halfStepRes;

        char waveStepsSize;
        char fullStepsSize;
        char halfStepsSize;

        char mode;
        char direction;
        char movementAmount;
        double moveDegrees;
        double sunAngle;
        int steps;
        char stepSeqIndex;

        char pins;
        } Motor;

/* Motor Functions */
Motor motor_init1(char pins)
{
        Motor motor = {
                {0x02, 0x08, 0x04, 0x01},       // waveSteps
                200,                               // steps
                1.8,                               // degree
                {0x03, 0x0A, 0x0C, 0x05},       // fullSteps
                200,                               // steps
                1.8,                               // degrees
                {0x03, 0x02, 0x0A, 0x08, 0x0C, 0x04, 0x05, 0x01}, // halfSteps
```

```
                400,                                    // steps
                0.9,                                    // degrees
                4,                                    // waveStepsSize
                4,                                    // fullStepsSize
                8,                                    // halfStepsSize
                FULLSTEPMODE,                           // mode (default = fullStepMode)
                IDLE,                                   // Idle position
                BYCYCLE,
                0.0,
                0.0,
                0,
                0,
                pins};                                    // PORTB pins 0-3

        DDRB |= pins; // set register B pins to high; configured as high
        return motor;
}

/* Motor Functions */
Motor motor_init4(char pins)
{
        Motor motor = {
                {0x08, 0x02, 0x04, 0x01},        // waveSteps
                200,                                    // steps
                1.8,                                    // degree
                {0x0A, 0x06, 0x05, 0x09},        // fullSteps
                200,                                    // steps
                1.8,                                    // degrees
                {0x08, 0x0A, 0x02, 0x06, 0x04, 0x05, 0x01, 0x09}, // halfSteps
                400,                                    // steps
                0.9,                                    // degrees
                4,                                    // waveStepsSize
                4,                                    // fullStepsSize
                8,                                    // halfStepsSize
                FULLSTEPMODE,                           // mode (default = fullStepMode)
                IDLE,                                   // Idle position
                BYCYCLE,
                0.0,
                0.0,
                0,
                0,
                pins};                                    // PORTB pins 0-3

        DDRB |= pins; // set register B pins to high; configured as high
        return motor;
```

```
}

void moveMotor(Motor *motor)
{
        /*
                Move motor one cycle in direction determined by
                position of sun.
        */
        int i = 0;
        int *steps;
        char numbStep = 0;
        double res = 0.0;
        int stepNumb = 0;

        switch(motor->mode)
        {
                case WAVESTEPMODE:
                        steps = &motor->waveSteps;
                        numbStep = motor->waveStepsSize;
                        res = motor->waveStepRes;
                        stepNumb = motor->waveStepNumb;
                        break;
                case FULLSTEPMODE:
                        steps = &motor->fullSteps;
                        numbStep = motor->fullStepsSize;
                        res = motor->fullStepRes;
                        stepNumb = motor->fullStepNumb;
                        break;
                case HALFSTEPMODE:
                        steps = &motor->halfSteps;
                        numbStep = motor->halfStepsSize;
                        res = motor->halfStepRes;
                        stepNumb = motor->halfStepNumb;
                        break;
        }

        if(motor->direction == GOCLOCKWISE)
        {
                if(motor->movementAmount == BYCYCLE)
                {
                        for(i=0; i<numbStep; i++)
                        {
                                // Turn off the stepper motor only
                                PORTB &= 0xF0;
                                // Cause the stepper motor to make a step
```

```
                        PORTB |= *(steps + i);
                        // Delay time between step and shutting off motor
                        _delay_ms(MOTORDELAY);

                        /* Increment sun angle */
                        motor->sunAngle = motor->sunAngle + res;
                        motor->steps = motor->steps + 1;

                        if(motor->steps > stepNumb - 1)
                        {
                                motor->steps = motor->steps - stepNumb;
                        }

                        if(motor->sunAngle > 360.0 - res)
                        {
                                motor->sunAngle = motor->sunAngle - 360;
                        }
                }
        }

        else if(motor->movementAmount == BYSTEP)
        {
                // Turn off the stepper motor only
                PORTB &= 0xF0;
                // Cause the stepper motor to make a step
                PORTB |= *(steps + motor->stepSeqIndex);
                // Delay time between step and shutting off motor
                _delay_ms(MOTORDELAY);
        }

        else
        {
                // do nothing
        }
}
else if(motor->direction == GOCOUNTERCLOCKWISE)
{
        if(motor->movementAmount == BYCYCLE)
        {
                for(i=numbStep-1; i>=0; i--)
                {
                        // Turn off the stepper motor only
                        PORTB &= 0xF0;
                        // Cause the stepper motor to make a step
                        PORTB |= *(steps + i);
```

```
                                // Delay time between step and shutting off motor
                                _delay_ms(MOTORDELAY);

                                /* Decrement sun angle */
                                motor->sunAngle = motor->sunAngle - res;
                                motor->steps = motor->steps - 1;

                                if(motor->steps < 0)
                                {
                                        motor->steps = motor->steps + stepNumb;
                                }

                                if(motor->sunAngle < 0)
                                {
                                        motor->sunAngle = motor->sunAngle + 360;
                                }
                        }
                }

                else if(motor->movementAmount == BYSTEP)
                {
                        // Turn off the stepper motor only
                        PORTB &= 0xF0;
                        // Cause the stepper motor to make a step
                        PORTB |= *(steps + motor->stepSeqIndex);
                        // Delay time between step and shutting off motor
                        _delay_ms(MOTORDELAY);
                }

                else
                {
                        // do nothing
                }
        }
        else
        {
                // Idle
        }
}
```