EEC496

Implementation of a Temperature Control Unit into a Gas Sensor Testing System

Peter Hua

48667059

Technical Supervisor: Jonas Flueckiger

Date Submitted: December 2, 2011

University of British Columbia

EECE496

Faculty of Applied Science

December 2, 2011

ABSTRACT

"Implementation of a Temperature Control Unit into a Gas Sensor Testing System"

By Peter Hua

The need for a temperature control unit arises from experiments in measuring material conductivity in the presence of different gases where the temperature is required to be controlled at a known value. The focus of this project was to build on the previous work done by a previous student who designed and fabricated the temperature control hardware by implementing the software control of the hardware. A microcontroller was programmed to interface with the temperature control circuit in addition to a graphical user interface that allows a user to adjust the temperature of the gas chamber.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

GLOSSARY

Arduino:	Hardware platform with a microcontroller, input/output pins and a software platform with C libraries that communicates with computer software via USB.
MATLAB:	Mathematical computing environment and programming language with a variety of libraries for data analysis, algorithm implementation, and creation of graphical user interfaces
PID:	Proportional-integral-derivative controller is a loop feedback mechanism used in control systems. The proportional, integral, and derivative parameters are adjusted to tune the system response.
PWM:	Pulse width modulation is a technique for getting an analog signal from a digital signal. By changing the amount of time a digital signal is on compared to the time it is off, a pulse width is achieved that corresponds to an analog value.
Wheatstone Bridge:	Used to measure an unknown resistance, using four resistors with two resistors in series with each other and parallel to another two resistors in the same configuration.

1.0 INTRODUCTION

This project presents an investigation into the design of a temperature control system for a gas sensor chamber. My objectives were to design and program a graphical user interface (GUI) in MATLAB* and the microcontroller of an Arduino* for software control of the temperature control hardware. This chamber will be used to measure material conductivities in the presence of different gasses and requires the temperature to be maintained at a known value. This project is a continuation of the work done by a previous co-op student, David Hung who did the design of the temperature control circuit and fabrication of a chamber. The report's contents discuss the work done in the the software implementation of temperature control. Some work was done in the design and fabrication of a metal enclosure for the circuit but its discussion is not within the scope of this report. The following sections of the report begins with an overview of the project, the designs and experiments related to the GUI and PID controller, a discussion of the software design, and the results of the project.

* This and all subsequent terms marked with an asterisk are defined in the Glossary, pp. vi

2.0 PROJECT OVIERVIEW

For this project, MATLAB was used to program the GUI while the Arduino was programmed to control the temperature control circuit using PID control. The two programs communicate synchronously to exchange temperature and PID data. On the hardware side of the system, a thermoelectric device (TE) relies on the Peltier effect to heat and cool the chamber and a thermistor provides temperature measurements by observing its resistance.

2.1 EQUIPMENT USED

A previous co-op student designed and fabricated a prototype of the temperature control circuit, which was improved and built by my technical supervisor Jonas Flueckiger. The development board chosen to interface with the temperature control circuit was the Arduino Duemilanove for its ease of development. The Arduino platform provides many libraries for C development at a high level and is programmable through a USB connection. MATLAB was used to program the GUI that controls the temperature control unit and other sensors used in the gas chamber. MATLAB has a variety of rich libraries for input/output communication as well as a GUI development library that allows for rapid prototyping. A Keithley SourceMeter to be used as a sensor in the system was also used to develop the GUI. Solidworks* was used to design an enclosure for the temperature control circuit and Arduino and was fabricated out of sheet metal.

2.2 TEMPERATRE CONTROL

There are different approaches in control theory to design a control algorithm that satisfies the requirements of a particular system. However, PID control is by far the most popular technique used in industry due to its effectiveness and simple implementation [1]. A distinct advantage of PID control is that it also allows a designer without any control theory knowledge to tune the dynamics of the system by changing the PID parameters. For these reasons, PID control was chosen as the control algorithm for this project.

2.3 SYSTEM DESCRIPTION

The overall system is a control loop that allows a user to control the gas chamber temperature. Refer to Figure 2.1 for the system overview diagram. The major components of the system are described below:

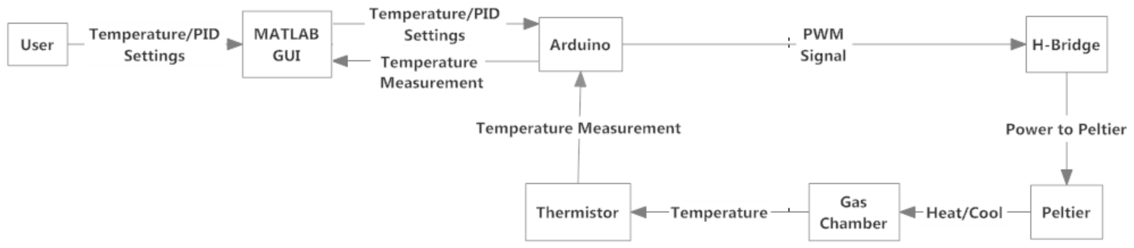


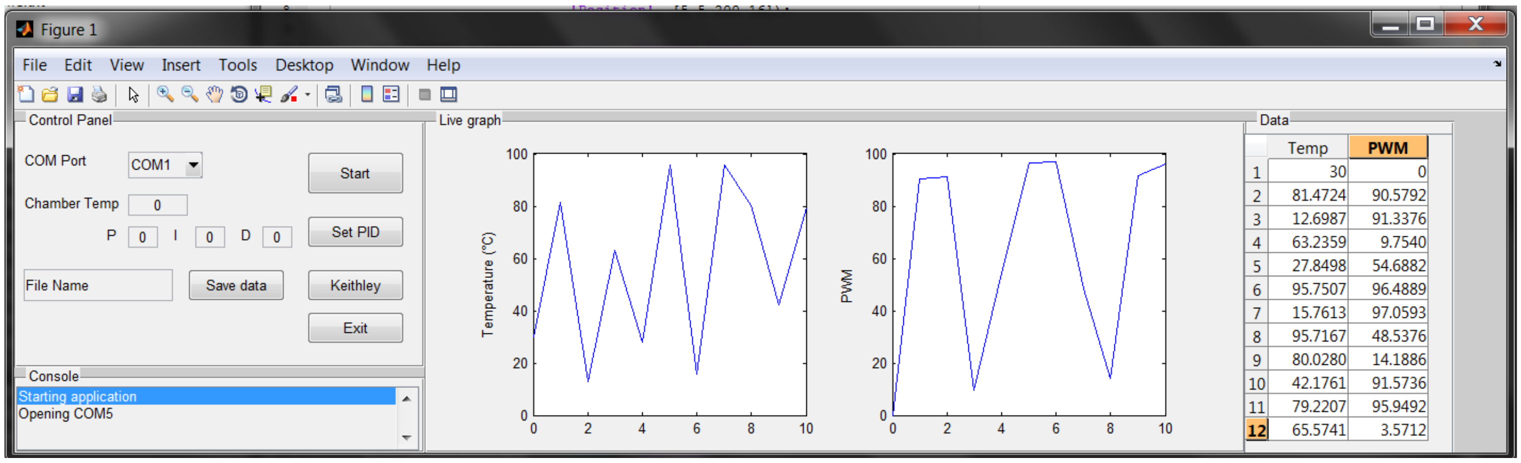
Figure 2.1: System Overview

2.3.1 MATLAB GUI

The MATLAB GUI allows the user to enter a desired temperature and the parameters for the PID controller, which are sent to the Arduino. The MATLAB program graphs the measured temperature and pulse width modulation (PWM) signals received from the

Arduino in real time. In addition, the MATLAB program communicates and plots the current, voltage, and resistance measurements of materials inside the gas chamber from a Keithley SourceMeter.

The main window of the GUI also serves as a launching point for the Keithley SourceMeter window, as well as other sensors that may be added to the system in the future. Refer to Figure 2.2 for a diagram of the MATLAB GUI.



COM Port COM1

Chamber Temp 0

P 0 I 0 D 0

Starting application
Opening COM5

	Temp	PWM
1	30	0
2	81.4724	90.5792
3	12.6987	91.3376
4	63.2359	9.7540
5	27.8498	54.6882
6	95.7507	96.4889
7	15.7613	97.0593
8	95.7167	48.5376
9	80.0280	14.1886
10	42.1761	91.5736
11	79.2207	95.9492
12	65.5741	3.5712

Figure 2.2: MATLAB GUI

2.3.2 Arduino

The input of the Arduino from the MATLAB program are the desired temperature and values of the proportional, integral, and derivative terms for PID control through a serial communication port. The temperature of the gas chamber is also an input to the Arduino from the thermistor. Using the values of the desired and actual temperature, the Arduino calculates an output using PID control and converts it into a PWM signal that is sent to the H-Bridge.

2.3.3 H-Bridge

The input of the H-Bridge is the PWM signal from the Arduino and its output is voltage to the TE device. Depending on the PWM signal, the polarity and magnitude of the voltage will vary accordingly. Refer to Appendix A for the circuit diagram with the H-Bridge.

2.3.4 Thermoelectric Device

The TE device is seated between the gas chamber and a heat sink. Applying voltage to the TE device will either heat or cool one of its faces depending on the direction of the voltage due to the Peltier effect [2]. During heating operation, the TE device will dissipate heat on the surface attached to the gas chamber while the other surface will remove heat, and vice versa for the cooling operation.

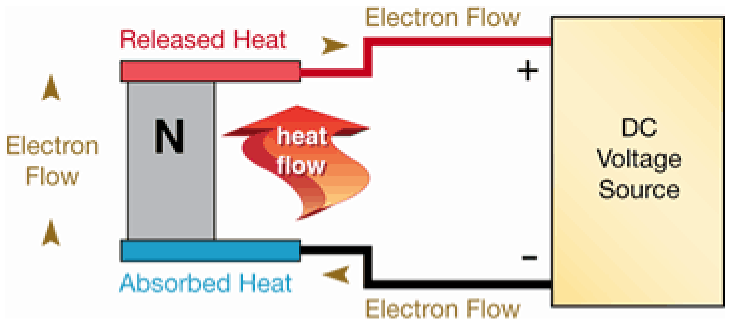


Figure 2.3: Peltier Effect

Source: http://www.tellurex.com/technology/peltier-faq.php

2.3.5 Thermistor

A thermistor is placed inside the gas chamber so that the Arduino monitors the temperature inside. The resistance of a thermistor varies according to the temperature, which can be determined indirectly by measuring the thermisor’s resistance and using a resistance to temperature conversion chart provided by the thermistor manufacturer.

2.3.6 Keithley SourceMeter

This instrument is controlled by the MATLAB program and sends resistance, current, and voltage measurements back through its GPIB port.

3.0 SOFTWARE DESIGN

3.1 MATLAB CODE

The following describes the flowchart in Figure 3.3 for the operation of the MATLAB temperature control program.

3.1.1 GUI Elements

When the temperature control program is started, some GUI initialization is done to draw all of the GUI components on the screen. The MATLAB GUI library divides GUI elements into a hierarchy that allows for logical organization of GUI elements. At the root of the hierarchy is the window, which is parent to panels that segment the window into separate regions. For the temperature control GUI, there are panels for the control panel, graph panel, table panel, and console panel. Within a panel, different types of GUI components can be created such as push buttons, drop down menus, and tables. Using panel organization, the control panel has push buttons that allow the user to set a desired temperature, exit the program, and save the recorded data.

Each GUI element is defined by its constructor function that defines its properties as well as a function callback that executes when it is activated. For example, the exit push button is created using the following constructor function where properties such as its position and font size are defined:

```
exit_pushbox = uicontrol('Parent', ctrl_panel,...
                'Style', 'pushbutton',...
                'Units', 'characters',...
                'FontSize', fontsize,...
                'String', 'Exit',...
                'Position', [39, 1, 13, 1.5],...
                'Callback', {@exit_pushbox_callback});
```

The following function is the callback where the behaviour of the pushbutton is defined and executes when the pushbutton is clicked. Since the temperature control code runs in a constant while loop, MATLAB will generate an interrupt and execute a GUI element's callback when it is activated before returning to main loop.

```
function exit_pushbox_callback(hObject, eventdata, handles)
%Push button behaviour
end
```

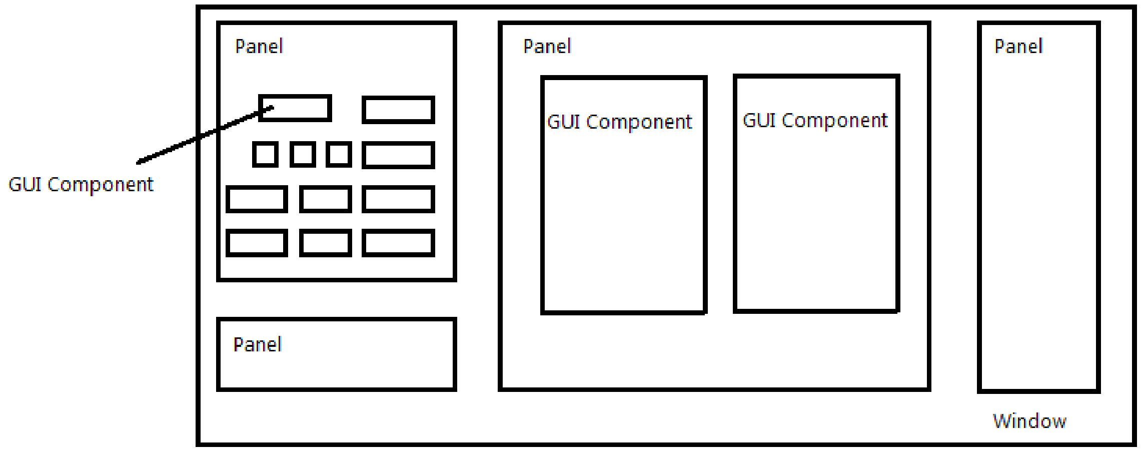


Figure 3.1: GUI Layout Hierarchy

3.1.2 Arduino Communication

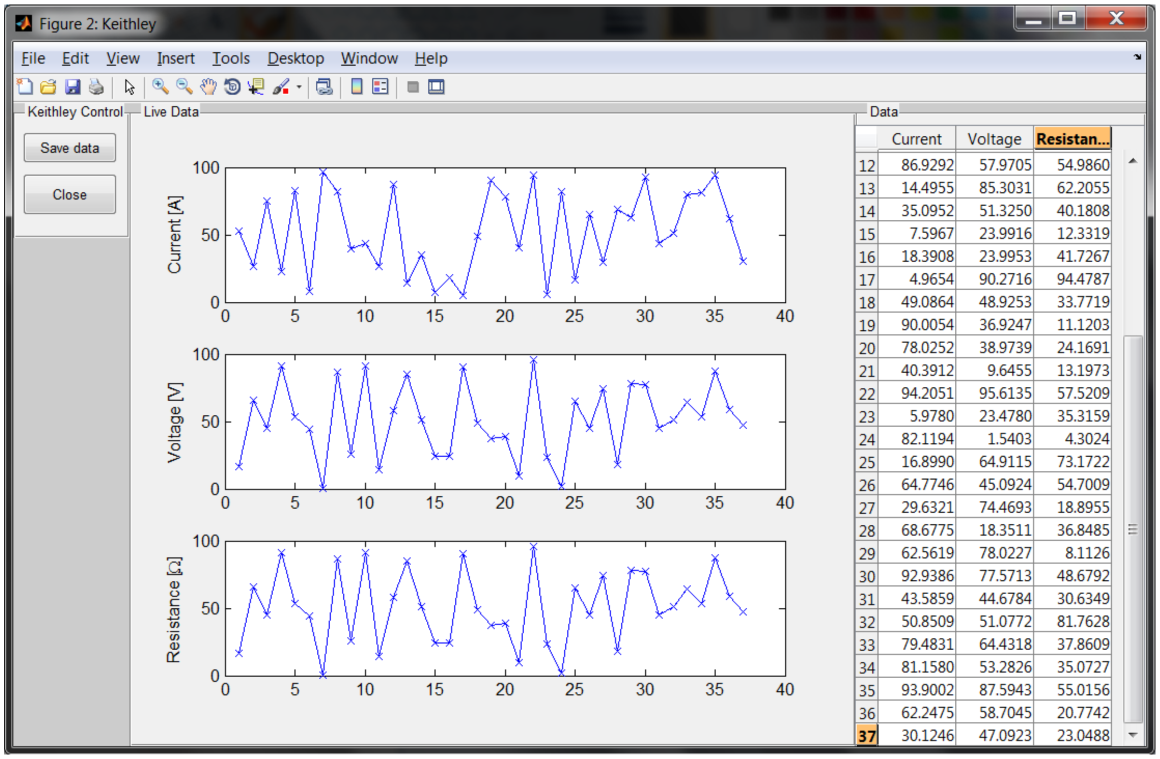
First the user must choose the COM port that the Arduino is connected to start using the temperature control program. The program then sends the temperature and PID values entered by the user to the Arduino. The program expects that the Arduino will send the measured temperature and PWM signal and waits until this data is available on the COM port buffer, which are plotted on the graphs.

The next operation is to check if the user has decided to run the Keithley SourceMeter, and if so displays a second window for Keithley control. My technical supervisor Jonas

Flueckiger wrote the MATLAB code that performs the Keithley measurements, which I integrated into the temperature control program.

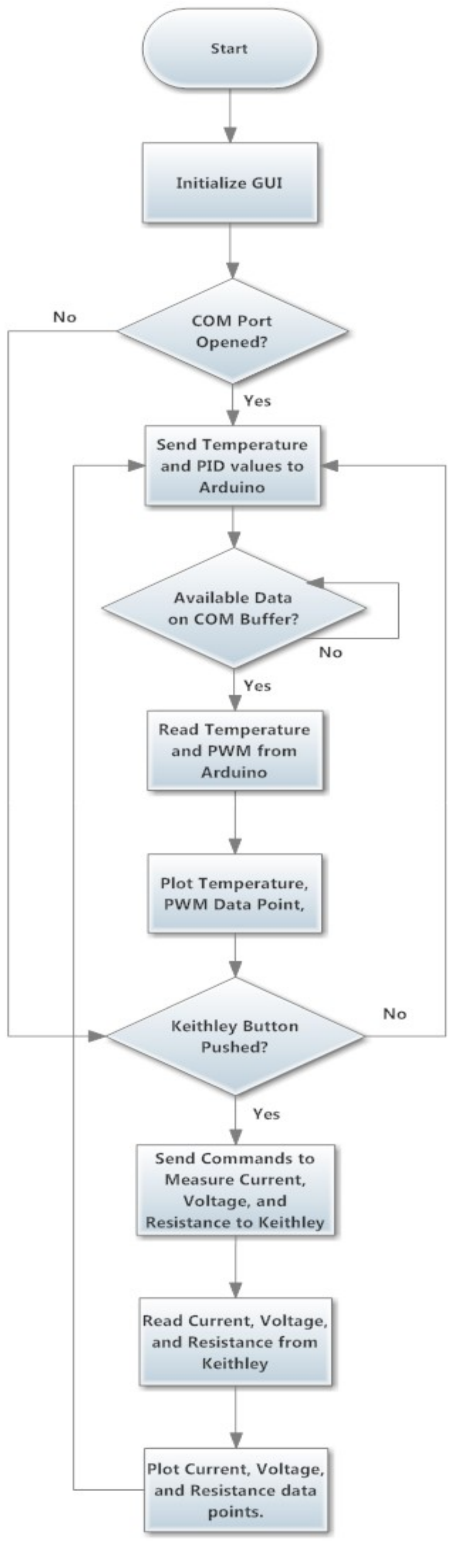
3.1.3 Keithley SourceMeter Communication

Communication with the Keithley SourceMeter is done through the GPIB port and measurements are recorded from both channels of the Keithley. The MATLAB code sends commands to measure the voltage, current, and resistance from both channels of the Keithley SourceMeter, which are stored on the GPIB buffer. These measurements are read and plotted onto the Keithley window graphs while the data table is updated. A short loop delay pauses the program before the next iteration of the loop begins.



	Current	Voltage	Resistan...
12	86.9292	57.9705	54.9860
13	14.4955	85.3031	62.2055
14	35.0952	51.3250	40.1808
15	7.5967	23.9916	12.3319
16	18.3908	23.9953	41.7267
17	4.9654	90.2716	94.4787
18	49.0864	48.9253	33.7719
19	90.0054	36.9247	11.1203
20	78.0252	38.9739	24.1691
21	40.3912	9.6455	13.1973
22	94.2051	95.6135	57.5209
23	5.9780	23.4780	35.3159
24	82.1194	1.5403	4.3024
25	16.8990	64.9115	73.1722
26	64.7746	45.0924	54.7009
27	29.6321	74.4693	18.8955
28	68.6775	18.3511	36.8485
29	62.5619	78.0227	8.1126
30	92.9386	77.5713	48.6792
31	43.5859	44.6784	30.6349
32	50.8509	51.0772	81.7628
33	79.4831	64.4318	37.8609
34	81.1580	53.2826	35.0727
35	93.9002	87.5943	55.0156
36	62.2475	58.7045	20.7742
37	30.1246	47.0923	23.0488

Figure 3.2: Keithley SourceMeter GUI


Start
Initialize GUI
COM Port Opened?
No
Yes
Send Temperature and PID values to Arduino
Available Data on COM Buffer?
No
Yes
Read Temperature and PWM from Arduino
Plot Temperature, PWM Data Point,
Keithley Button Pushed?
No
Yes
Send Commands to Measure Current, Voltage, and Resistance to Keithley
Read Current, Voltage, and Resistance from Keithley
Plot Current, Voltage, and Resistance data points.

3.3: MATLAB Program Flowchart

3.2 ARDUINO CODE

The following describes in detail the flowchart in Figure 3.4, which outlines the operation of the Arduino temperature control code.

3.2.1 MATLAB Communication

First the COM port is opened and the set temperature and the values for PID control sent from the MATLAB program are read. The program waits for data to be available on the serial port before reading in each value. When the temperature is calculated from the measured voltage as described in the following sections, the Arduino sends this information back to the MATLAB program. In addition, the PWM signal calculated from PID control is also sent to MATLAB.

3.2.2 Voltage Measurement

The voltage is read from the temperature control circuit, which is used to calculate the resistance of the thermistor that is converted into temperature. Refer to Appendix A for the temperature control circuit.

3.2.3 Temperature Conversion

In the temperature sensor circuitry, the thermistor is placed in a Wheatstone bridge* where the voltage across the bridge is amplified using a differential amplifier. The differential amplifier is referenced to 2.5 V since the Arduino reads voltages between 0-5 V and maps them between 0-1024. Using the measured value of the amplifier output and the reference voltage of the differential amplifier, the resistance across the Wheatstone bridge is calculated by the Arduino.

The differential amplifier output referenced to ground, V_o is calculated from the actual output, V_{o+} and the reference voltage of 2.5 V, V_{o-}.

$$V_o = (V_{o+} - V_{o-}) * \frac{5}{1024} \quad (1)$$

The output of the differential amplifier divided by the gain calculates the input of the differential amplifier, or the voltage across the Wheatstone bridge. The gain of the amplifier, G is set with an adjustable resistor, R_{adj2} such that the output does not exceed 5V within the temperature range of the gas chamber. The input voltage range of the Arduino is the limiter as any voltages exceeding 5V will be aliased to 5V.

$$G = 1 + \frac{50k\Omega}{R_{adj2}} \quad (2)$$

$$V_i = V_o/G \quad (3)$$

Since the input of the differential amplifier is the voltage across the Wheatstone bridge, the voltage across the thermistor is calculated by subtracting the amplifier input from the voltage across the adjustable resistor. The adjustable resistance, R_{adj} is set such that the bridge is balanced at room temperature, meaning that the voltage across the bridge is zero.

$$V_{therm} = v_{adj} - V_i \quad (4)$$

The voltage across the thermistor is then used to calculate the thermistor's resistance.

$$R_{therm} = V_{therm} * \frac{R3}{5 - V_{therm}} \quad (5)$$

Finally, the resistance is converted into temperature using the resistance to temperature conversion chart supplied from the manufacturer, see Appendix B.

3.2.4 PID Control

The values for P, I, and D are adjusted to control the response of the system and are supplied by the user. The formula to calculate the system output at time *t*, u*(t)* is the sum of the proportional, derivative, and integral terms [3]. The proportional term is calculated from the value for *P* set by the user and the error at time *t*, *e(t)* which is the difference between the desired system output and the actual system output. In this case, the desired system output is the temperature set by the user and the actual system output is the temperature measured. The integral error is calculated from the *I* value and the sum of all errors while the derivative term is the derivative of the error multiplied by the *D* value.

$$u(t) = K_p * e(t) + K_i \int e(t) + K_D * \frac{de(t)}{dt} \quad (6)$$

3.2.5 PWM Output

PWM is a technique to obtain an analog signal using a digital voltage by varying the amount of time a signal is on (5 V) relative to the time it is off (0 V). This allows a cooling or heating signal to be sent to the TE device that can vary in magnitude. Sending a PWM signal to one of the two H-Bridges will determine the polarity of the output voltage as seen by the TE device.

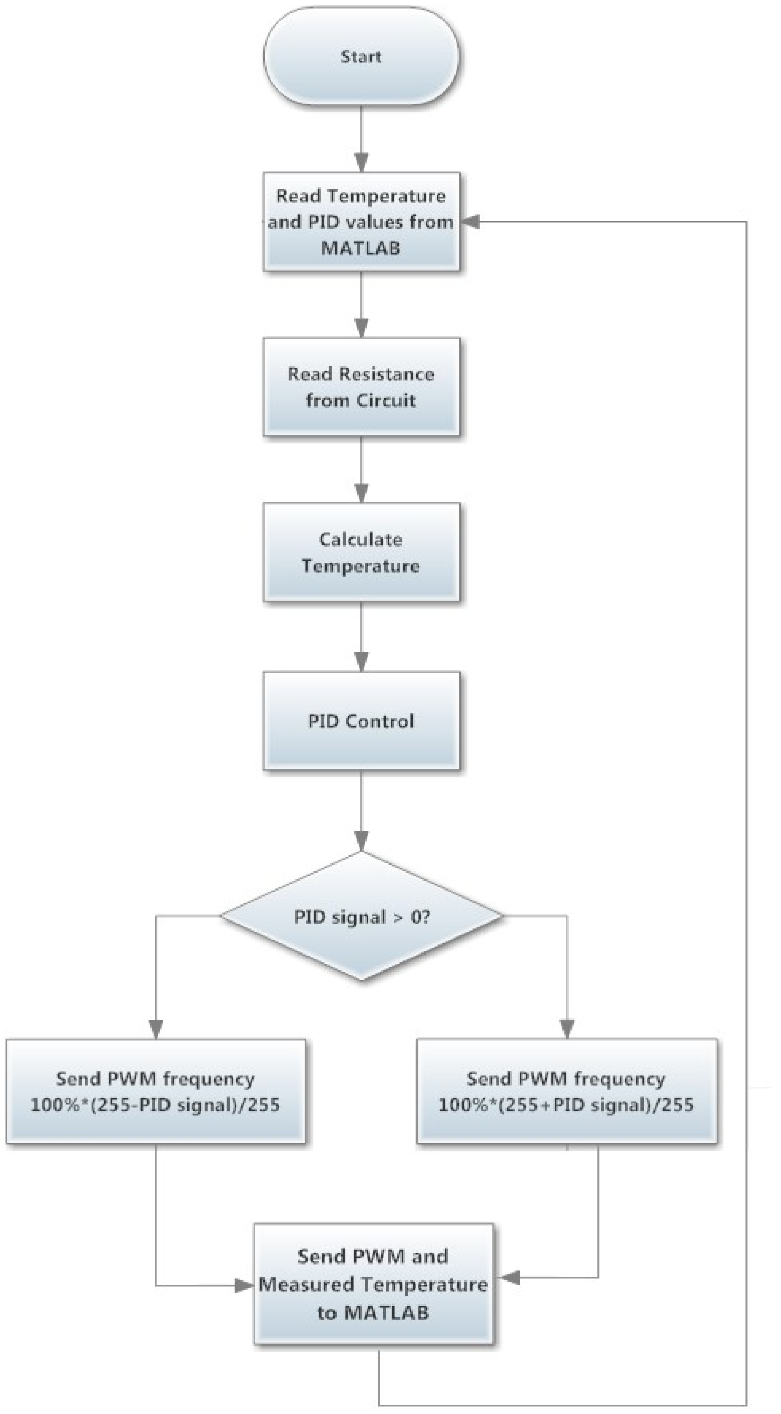

Start
Read Temperature and PID values from MATLAB
Read Resistance from Circuit
Calculate Temperature
PID Control
PID signal > 0?
Send PWM frequency 100%*(255-PID signal)/255
Send PWM frequency 100%*(255+PID signal)/255
Send PWM and Measured Temperature to MATLAB

Figure 3.4: Arduino Program Flowchart

4.0 RESULTS

The goals of the project to program a GUI and PID for a temperature control system were achieved. Some difficulties were met during implementation, which are discussed in the following sections as well as suggestions for future development.

4.1 EVALUATION

The objectives outlined at the outset of the project were achieved. A MATLAB GUI that allows a user to control the temperature inside of a small gas chamber integrated with conductivity measurements from a Keithley SourceMeter was programmed. An Arduino was programmed to interface with the MATLAB program and to control a temperature control circuit using PID control.

An environment chamber for an inverted microscope fabricated by a co-op student from a previous project was used to test the overall system. The Arduino code was also tested using this chamber and an attempt was made to manually tune the PID controller. After testing several values, an acceptable system response was achieved using P, I, D values of 60, 1, and 3, respectively. A step input setting the chamber temperature to 59 degrees Celsius using these PID values results in a system response with an overshoot of 1 degrees and a settling time of 400 seconds with minimal setting error. Figure 4.1 shows the system step response and Figure 4.2 shows the corresponding PWM frequencies calculated and sent to the H-Bridge from the Arduino. Note that in Figure 4.2 a negative PWM frequency corresponds to frequencies being sent to the H-Bridge to reverse the current to the TE device, thus cooling the chamber.

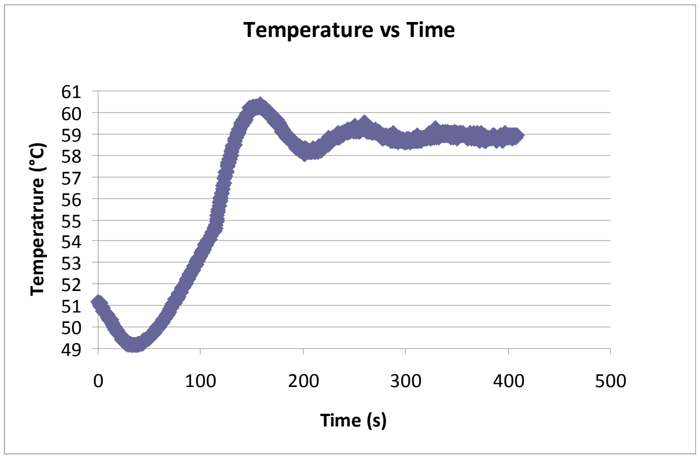


Figure 4.1: System Response to a Step Input

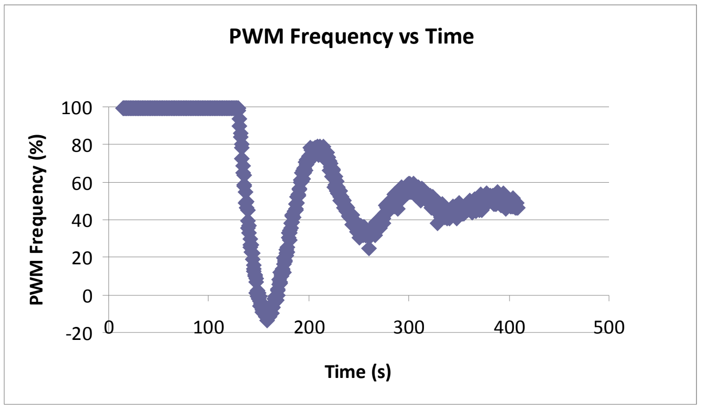


Figure 4.2: System Output to a Step Input

4.2 DIFFICULTIES

There were many challenges encountered through the course of this project. Difficulties were encountered during development of the Arduino since it is impossible to debug code as one would normally do on code executed on a personal computer. Most compilers on a computer offer debug tools that allow the user to set breakpoints and step through the code. The ability to watch variables while stepping through code is also extremely useful during debugging. Unfortunately, none of these features are available to the developer programming code that runs on a microcontroller. This problem was somewhat circumvented by sending debug information from the Arduino to the MATLAB program and viewing it in MATLAB.

Other obstacles encountered during development of the GUI were due to the limitations of the MATLAB GUI library. For example, there is no way of controlling the scroll bar with the GUI library. When text is added to a table such that the amount of text exceeds the height of the GUI component, the cursor will automatically move to the first row. As a result, the GUI component always scrolls to the first line whenever new content is added, which occurs when data is sampled every half second. Workarounds were found to circumvent these issues through experimentation and online research. In the case of the scroll bar control functionality, the desired scroll bar behaviour was achieved by manipulating the GUI component through MATLAB-Java code.

Another challenge was taking MATLAB code that was written to control a Keithley SourceMeter and integrating it into the temperature control program. There was some difficulty in synchronizing communication with both the Arduino and Keithley and defining the behaviour of their GUI components such that they function independent of each other.

4.3 SUGGESTIONS FOR FUTURE DEVELOPMENT

Currently the MATLAB program is able to communicate with two Keithley SourceMeters and was implemented by duplicating the code for one Keithley. A more scalable approach to improve the program is to create the Keithley GPID objects in an array and to perform the communication inside of a loop. This will allow the user to define the number of Keithley SourceMeters being used instead of having the functionality of two Keithleys hard coded into the program. An added benefit is that it reduces the size of the code since it loops instead of stepping through redundant code.

To improve the accuracy of the temperature measurements, the Steinhart–Hart equation can be used to model the resistance of the thermistor according to temperature instead of using a conversion table. The following equation requires three coefficients to model a thermistor which can be found by measuring three data points for resistance and temperature:

$$\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$$

If the functionality of the MATLAB GUI was extended to allow the user to send these coefficients to the Arduino, the thermistor can be replaced without having to modify the Arduino code.

The current PID implementation for the Arduino stores the errors to calculate the integral term inside an array of length defined by the user. This is to combat a phenomenon known as integral windup where the system takes a long time to recover when a large step input is applied [4]. A large step input will cause the system output to saturate and the integral error to accumulate until the system overshoots enough such that the error is unwound and the system output decreases toward the set point. This phenomenon occurs in non-ideal control systems because the system output is limited to some maximum and minimum. In this particular system, the system limits are 100% PWM frequency with polarity to heat the chamber and 100% PWM frequency with polarity to cool the chamber. It is impossible for the PID to drive a signal larger than a 100% PWM signal but the PID output will continue to increase due to the increasing integral error.

A common anti-windup technique known as tracking modifies the PID transfer function by recomputing the integral error when the system output saturates. The implementation of tracking is relatively simple and uses the following formula during tracking operation (when the output is saturated) to re-calculate the integral error, I_{error} from the saturated system output $u(t)_{limited}$, calculated system output $u(t)$, and a constant T_t:

$$I_{error} = I_{error} + \frac{u(t)_{limited} - u(t)}{T_t}$$

Normal operation occurs when the system is not saturated, meaning that the PID output is less than the maximum system output. When the system reaches saturation, a second feedback path will re-calculate the integral error by subtracting the calculated system output from the actual system output and dividing by a large constant chosen by the user. This prevents the integral error from increasing to a large value and thus prevents windup.

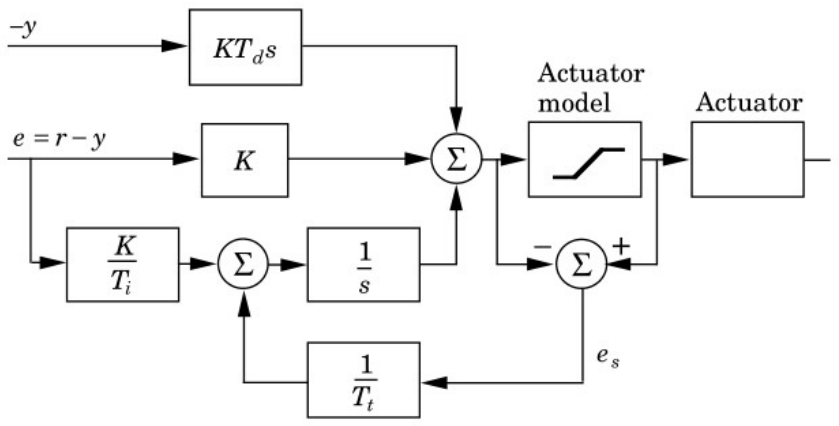


Figure 4.3: Anti-windup Tracking

5.0 CONCLUSION

This report investigated an implementation of a temperature control unit for a gas chamber. The existing hardware system designed and fabricated by a previous student consists of an H-Bridge to drive a TE device, which has the ability to heat or cool the chamber and a sensor to measure the temperature. Using the Arduino microcontroller, a program was written to read from the temperature sensor and to control the system using PID control. A graphical user interface was programmed in MATLAB to allow a user to control the temperature and the PID parameters by interfacing with the Arduino.

The project was complement successfully with all of the features in the project specifications. Some difficulties were encountered related to the capabilities of the MATLAB GUI library and the development on the microcontroller, but workarounds were found to avoid these problems. For future students with an interest in control theory, anti-windup techniques can be implemented in the Arduino code to improve the PID.

6.0 REFERENCES

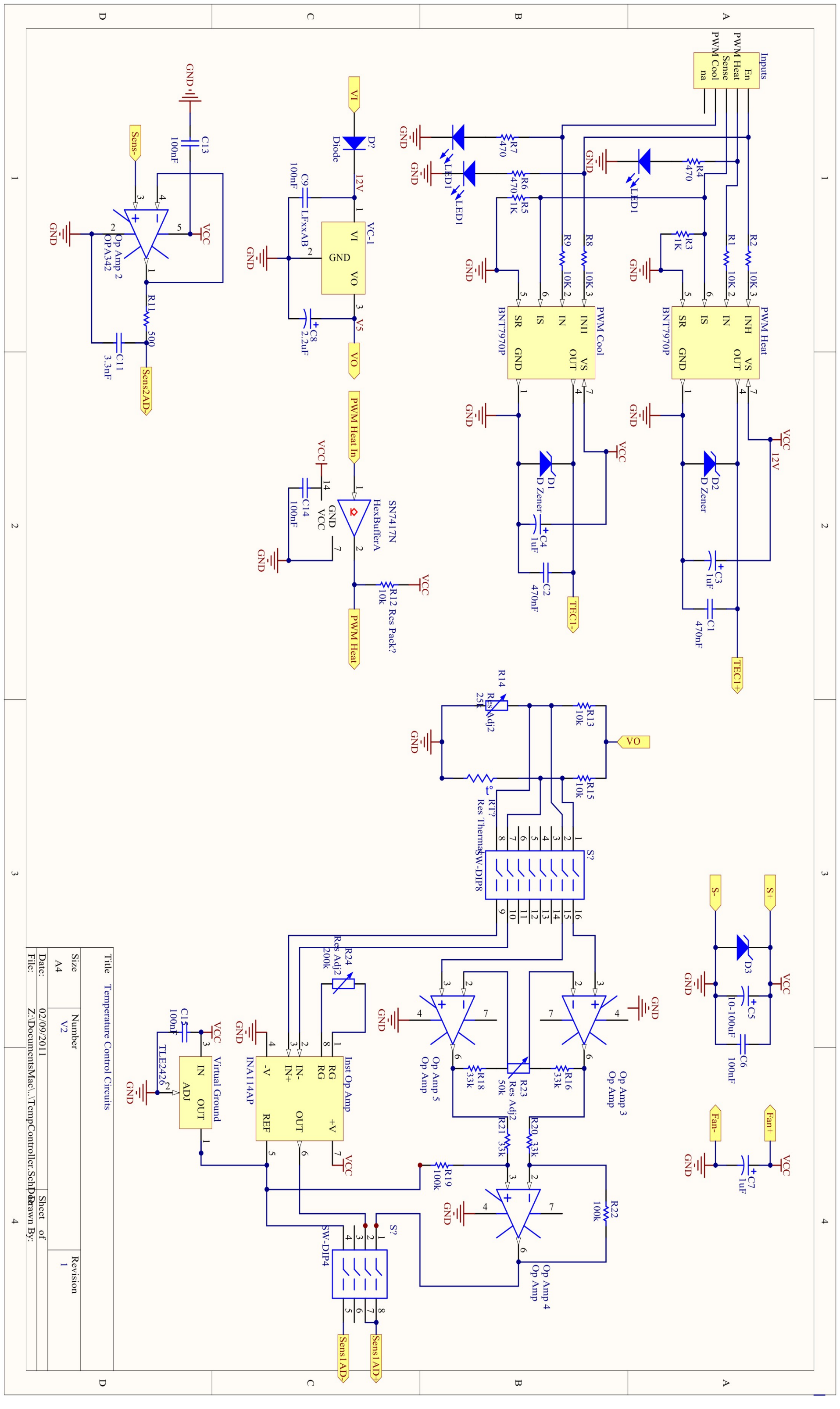
[1] Tellurex. “Frequently Asked Questions About Our Cooling And Heating Technology.” [online], Tellurex, 2011 [October 2, 2011], Available: <http://www.tellurex.com/technology/peltier-faq.php>

[2] National Instruments. “Improving PID Controller Performance.” [online], National Instruments, Dec. 3, 2009 [October 16, 2011], Available: <http://zone.ni.com/devzone/cda/tut/p/id/7438>

[3] R.C. Dorf and R.H. Bishop, “The Root Locus Method,” *Modern Control Systems*, 12th ed., New Jersey: Prentice Hall, 2008, 480.

[4] K.J. Astrom and R.M. Murray, “PID Control,” *Feedback Systems: An Introduction for Scientists and Engineers*, 1st ed., New Jersey: Princeton University Press, 2008, 306-308.

APPENDIX A – Circuit Diagram



Appendix B – Resistance to Temperature Conversion Chart

EPCOS

Temperature measurement	B57861S
Miniature sensors with bendable wires	**S861**

R/T characteristics

R/T No.	**8016**		**8018**	
T (°C)	$B_{25/100}$ = 3988 K		$B_{25/100}$ = 3964 K	
	R_T/R_{25}	α (%/K)	R_T/R_{25}	α (%/K)
−55.0	96.3	7.4	–	–
−50.0	67.01	7.2	–	–
−45.0	47.17	6.9	–	–
−40.0	33.65	6.7	30.24	6.3
−35.0	24.26	6.4	22.1	6.1
−30.0	17.7	6.2	16.32	5.9
−25.0	13.04	6.0	12.17	5.8
−20.0	9.707	5.8	9.153	5.6
−15.0	7.293	5.6	6.945	5.4
−10.0	5.533	5.5	5.313	5.2
−5.0	4.232	5.3	4.097	5.1
0.0	3.265	5.1	3.183	4.9
5.0	2.539	5.0	2.491	4.8
10.0	1.99	4.8	1.963	4.7
15.0	1.571	4.7	1.557	4.6
20.0	1.249	4.5	1.244	4.4
25.0	1.0000	4.4	1.0000	4.3
30.0	0.8057	4.3	0.8083	4.2
35.0	0.6531	4.1	0.6572	4.1
40.0	0.5327	4.0	0.5373	4.0
45.0	0.4369	3.9	0.4418	3.9
50.0	0.3603	3.8	0.365	3.7
55.0	0.2986	3.7	0.303	3.7
60.0	0.2488	3.6	0.2527	3.6
65.0	0.2083	3.5	0.2118	3.5
70.0	0.1752	3.4	0.1783	3.4
75.0	0.1481	3.3	0.1508	3.3
80.0	0.1258	3.2	0.128	3.2
85.0	0.1072	3.2	0.1091	3.2
90.0	0.09177	3.1	0.0933	3.1
95.0	0.07885	3.0	0.08016	3.0
100.0	0.068	2.9	0.0691	2.9
105.0	0.05886	2.9	0.05974	2.9
110.0	0.05112	2.8	0.05183	2.8
115.0	0.04454	2.7	0.04512	2.8
120.0	0.03893	2.6	0.0394	2.7
125.0	0.03417	2.6	0.0345	2.6
130.0	0.03009	2.5	0.03032	2.6
135.0	0.02654	2.5	0.02672	2.5
140.0	0.02348	2.4	0.02361	2.5
145.0	0.02083	2.4	0.02091	2.4
150.0	0.01853	2.3	0.01857	2.4
155.0	0.01653	2.3	0.016537	2.3

APPENDIX C – MATLAB Code

```
function temp_control()

fontsize = 8;

%--------------------------GUI PANELS-------------------------------------

    %%%%%%%%%%%%%%%%%%%%%%%%Temperature Control%%%%%%%%%%%%%%%%%%%%%%%%
    window = figure('Units', 'characters',...
                        'Visible', 'on',...
                        'Position', [5 5 200 16]);

    ctrl_panel = uipanel('Title', 'Control Panel',...
                        'Units', 'characters',...
                        'Position', [0 4 55 12],...
                        'Parent', window);

    graph_panel = uipanel('Title', 'Live graph',...
                         'Units', 'characters',...
                         'Position', [55 0 110 16],...
                         'Parent', window);

    data_panel = uipanel('Title', 'Data',...
                         'Units', 'characters',...
                         'Position', [165 0 28 16],...
                         'Parent', window);

    console_panel = uipanel('Title', 'Console',...
                         'Units', 'characters',...
                         'Position', [0 0 55 4],...
                         'Parent', window);

    %Calculate the correct position for the  data table workaround
    %An old version of uitable is used, which does not support 'Position' in characters so it must be converted into pixels.
    size_characters = get(window, 'Position');
    set(gcf, 'Units', 'Pixels');
    size_pixels = get(window, 'Position');
    char_to_pix = size_pixels(3:4)./size_characters(3:4);

    graph_panel_pos = get(graph_panel, 'Position');

    data_table_pos_x = char_to_pix(1)*(graph_panel_pos(1) + graph_panel_pos(3));
    data_table_pos_y = 0;
    data_table_pos_h = char_to_pix(2)*(graph_panel_pos(4)-1);
    data_table_pos_w = char_to_pix(1)*28;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%Keithley%%%%%%%%%%%%%%%%%%%%%%%%

    window2 = figure('name', 'Keithley',...
                        'Units', 'characters',...
                        'Visible', 'off',...
                        'Position', [5 1 156 55]);

    keithley_graph_panel_A = uipanel('Title', 'Channel A',...
                         'Units', 'characters',...
                         'Position', [16 27 100 27],...
                         'Parent', window2);

    keithley_graph_panel_B = uipanel('Title', 'Channel B',...
                         'Units', 'characters',...
                         'Position', [16 0 100 27],...
                         'Parent', window2);
    keithley_ctrl_panel = uipanel('Title', 'Keithley Control',...
                        'Units', 'characters',...
                        'Position', [0 24.5 16 6.5],...
                        'Parent', window2);

    keithley_data_panel = uipanel('Title', 'Data',...
                        'Units', 'characters',...
                        'Position', [116 0 40 27],...
                        'Parent', window2);

    %Calculate the correct position for the data table workaround
    %An old version of uitable is used, which does not support 'Position' in characters so it must be converted into pixels.
    keithley_graph_panel_pos = get(keithley_graph_panel_A, 'Position');

    keithley_data_table_pos_x = char_to_pix(1)*(keithley_graph_panel_pos(1) + keithley_graph_panel_pos(3));
    keithley_data_table_pos_y = char_to_pix(2)*keithley_graph_panel_pos(2);
    keithley_data_table_pos_h = char_to_pix(2)*(keithley_graph_panel_pos(4)-1);
    keithley_data_table_pos_w = char_to_pix(1)*40;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%keithley2%%%%%%%%%%%%%%%%%%%%%%%%%

    window3 = figure('name', 'keithley2',...
                       'Units', 'characters',...
                       'Visible', 'off',...
                       'Position', [5 1 156 55]);

    keithley_graph_panel_A2 = uipanel('Title', 'Channel A',...
                        'Units', 'characters',...
                        'Position', [16 27 100 27],...
                        'Parent', window3);

    keithley_graph_panel_B2 = uipanel('Title', 'Channel B',...
                        'Units', 'characters',...
                        'Position', [16 0 100 27],...
                        'Parent', window3);

    keithley_ctrl_panel2 = uipanel('Title', 'keithley2 Control',...
                        'Units', 'characters',...
                        'Position', [0 24.5 16 6.5],...
                        'Parent', window3);

    keithley_data_panel2 = uipanel('Title', 'Data',...
                        'Units', 'characters',...
                        'Position', [116 0 40 27],...
                        'Parent', window3);



    %Calculate the correct position for the data table workaround
    %An old version of uitable is used, which does not support 'Position' in characters so it must be converted into pixels.
    keithley_graph_panel_pos2 = get(keithley_graph_panel_A2, 'Position');

    keithley_data_table_pos_x2 = char_to_pix(1)*(keithley_graph_panel_pos2(1) + keithley_graph_panel_pos2(3));
    keithley_data_table_pos_y2 = char_to_pix(2)*keithley_graph_panel_pos2(2);
    keithley_data_table_pos_h2 = char_to_pix(2)*(keithley_graph_panel_pos2(4)-1);
    keithley_data_table_pos_w2 = char_to_pix(1)*40;

%--------------------------------------------------------------------------


%--------------------------GUI ELEMENTS------------------------------------

    %%%%%%%%%%%%%%%%%%%%%%%Temperature Control%%%%%%%%%%%%%%%%%%%%%%%
    ports_dropdown_label = uicontrol('Parent', ctrl_panel,...
                        'Units', 'characters',...
                        'FontSize', fontsize,...
                        'Style', 'text',...
                        'String', 'COM Port',...
                        'HorizontalAlignment', 'left',...
                        'Position', [1, 9, 10, 1]);

    %Get the avaliable COM ports. Note that MATLAB generates this on startup, so the Arduino must be connected before
%MATLAB is started.
    com_info = instrhwinfo('serial');
    com_avail = com_info.SerialPorts;

    ports_dropdown = uicontrol('Parent', ctrl_panel,...
                        'Units', 'characters',...
                        'FontSize', fontsize,...
                        'Style', 'popupmenu',...
                        'String', com_avail,...
                        'Position', [15, 9, 10, 1],...
                        'Callback', {@ports_dropdown_callback});

    temp_editbox_label = uicontrol('Parent', ctrl_panel,...
                        'Units', 'characters',...
                        'FontSize', fontsize,...
                        'Style', 'text',...
                        'String', 'Chamber Temp',...
                        'HorizontalAlignment', 'left',...
                        'Position', [1, 6.5, 13, 1.5]);

    temp_editbox = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'edit',...
                         'Units', 'characters',...
                         'String', '0',...
                         'Position', [15, 7, 8, 1],...
                         'Callback', {@temp_editbox_callback});

    start_stop_pushbox = uicontrol('Parent', ctrl_panel,...
                         'Style', 'pushbutton',...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'String', 'Start',...
                         'Position', [39, 8, 13, 2],...
                         'Callback', {@start_stop_pushbox_callback});

    p_editbox_label = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Text',...
                         'Units', 'characters',...
                         'String', 'P',...
                         'HorizontalAlignment', 'left',...
                         'Position', [12, 5, 2, 1.5]);

    p_editbox = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Edit',...
                         'Units', 'characters',...
                         'String', '0',...
                         'Position', [15, 5.5, 4, 1],...
                         'Callback', {@p_editbox_callback});

    i_editbox_label = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Text',...
                         'Units', 'characters',...
                         'String', 'I',...
                         'HorizontalAlignment', 'left',...
                         'Position', [21, 5, 1, 1.5]);

    i_editbox = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Edit',...
                         'Units', 'characters',...
                         'String', '0',...
                         'Position', [24, 5.5, 4, 1],...
                         'Callback', {@i_editbox_callback});

    d_editbox_label = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Text',...
                         'Units', 'characters',...
                         'String', 'D',...
                         'HorizontalAlignment', 'left',...
                         'Position', [30, 5, 2, 1.5]);

    d_editbox = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'Edit',...
                         'Units', 'characters',...
                         'String', '0',...
                         'Position', [33, 5.5, 4, 1],...
                         'Callback', {@d_editbox_callback});

    pid_set_pushbox = uicontrol('Parent', ctrl_panel,...
                         'Style', 'pushbutton',...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'String', 'Set PID',...
                         'Position', [39, 5.5, 13, 1.5],...
                         'Callback', {@pid_set_pushbox_callback});

    exit_pushbox = uicontrol('Parent', ctrl_panel,...
                         'Style', 'pushbutton',...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'String', 'Exit',...
                         'Position', [39, 1, 13, 1.5],...
                         'Callback', {@exit_pushbox_callback});

    filename_editbox_label = uicontrol('Parent', ctrl_panel,...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'Style', 'edit',...
                         'Units', 'characters',...
                         'String', 'File Name',...
                         'HorizontalAlignment', 'left',...
                         'Position', [1, 3, 20, 1.5]);

    save_data_pushbox = uicontrol('Parent', ctrl_panel,...
                         'Style', 'pushbutton',...
                         'Units', 'characters',...
                        'FontSize', fontsize,...
                         'String', 'Save data',...
```

```
                        'Position', [23, 3, 13, 1.5],...
                        'Callback', {@save_data_pushbox_callback});

    console_editbox = uicontrol('Parent', console_panel,...
                          'Style', 'listbox',...
                          'Max', 2,...
                          'HorizontalAlignment', 'left',...
                          'Units', 'characters',...
                          'FontSize', 8,...
                          'Position', [0, 0, 54, 3],...
                          'Callback', {@console_editbox_callback});

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Keithley%%%%%%%%%%%%%%%%%%%%%%%%%%
    keithley_pushbox = uicontrol('Parent', ctrl_panel,...
                        'Style', 'pushbutton',...
                        'Units', 'characters',...
                        'FontSize', fontsize,...
                        'String', 'Keithley',...
                        'Position', [0, 1, 13, 1.5],...
                        'Callback', {@keithley_pushbox_callback});

    keithley_close_pushbox = uicontrol('Parent', keithley_ctrl_panel,...
                          'Style', 'pushbutton',...
                          'Units', 'characters',...
                          'FontSize', fontsize,...
                          'String', 'Close',...
                          'Position', [1, 1, 13, 2],...
                          'Callback', {@keithley_close_pushbox_callback});

    keithley_save_data_pushbox = uicontrol('Parent', keithley_ctrl_panel,...
                          'Style', 'pushbutton',...
                          'Units', 'characters',...
                          'FontSize', fontsize,...
                          'String', 'Save data',...
                          'Position', [1, 3.5, 13, 1.5],...
                          'Callback', {@keithley_save_data_pushbox_callback});

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%keithley2%%%%%%%%%%%%%%%%%%%%%%%%%%
    keithley_pushbox2 = uicontrol('Parent', ctrl_panel,...
                        'Style', 'pushbutton',...
                        'Units', 'characters',...
                        'FontSize', fontsize,...
                        'String', 'Keithley2',...
                        'Position', [15, 1, 13, 1.5],...
                        'Callback', {@keithley_pushbox_callback2});

    keithley_close_pushbox2 = uicontrol('Parent', keithley_ctrl_panel2,...
                          'Style', 'pushbutton',...
                          'Units', 'characters',...
                          'FontSize', fontsize,...
                          'String', 'Close',...
                          'Position', [1, 1, 13, 2],...
                          'Callback', {@keithley_close_pushbox_callback2});

    keithley_save_data_pushbox2 = uicontrol('Parent', keithley_ctrl_panel2,...
                          'Style', 'pushbutton',...
                          'Units', 'characters',...
                          'FontSize', fontsize,...
                          'String', 'Save data',...
                          'Position', [1, 3.5, 13, 1.5],...
                          'Callback', {@keithley_save_data_pushbox_callback2});
%--------------------------------------------------------------------------


%----------------------------GUI VARIABLES---------------------------------
```

```
%%%%%%%%%%%%%%%%%%%%%%%%Temperature Control%%%%%%%%%%%%%%%%%%%%%%%%%
%Array storing the user input temperatures
global set_temp_history
set_temp_history = 0;

%Serial port being used
global port;

%P value set by user
global p_value;
p_value = 0;

%I value set by user
global i_value;
i_value = 0;

%D value set by user
global d_value;
d_value = 0;

%Exit flag signalling while loop to exit
global exit;
exit = 0;

%Console output history
global console_history
console_history = 'Starting application';
global console_i;
console_i = 2;

%Port status
global port_flag;
port_flag = 0;

%Start/Stop
global start;
start = 0; %OFF

%Console row length
global console_size;
console_size = 1;

%Temp control table row length
global data_table_size;
data_table_size = 1;

%Loop every...
global update;
update = 1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Keithley%%%%%%%%%%%%%%%%%%%%%%%%%
%Keithley time index
global n;
n = 1;

%Flag to setup communication with Keithley
%If set to 1, setup has already been done
global keithley;
keithley = 0; %OFF

%Flag for the Keithley window
global keithley_start;
keithley_start = 0;
```

```
    %Flag for Keithley window
    global keithley_close;
    keithley_close = 0;

    %Keithley Table row length
    global keithley_data_table_size;
    keithley_data_table_size = 0;

    %Initialization for Keithley graph
    Ilevel=1E-6;  % Current in A
    Vlevel=3;
    VlevelC=num2str(Vlevel);
    IlevelC=num2str(Ilevel);
    length= 2000;
    fC=num2str(update);
    gas='EtOH';
    filename= ['S1-' VlevelC 'A-f' fC 'Hz' '-Gas' gas '-2'];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%keithley2%%%%%%%%%%%%%%%%%%%%%%%%%%
    %keithley2 time index
    global n2;
    n2 = 1;

    %Flag to setup communication with keithley2
    %If set to 1, setup has already been done
    global keithley2;
    keithley2 = 0; %OFF

    %Flag for the keithley2 window
    global keithley_start2;
    keithley_start2 = 0;

    %Flag for keithley2 window
    global keithley_close2;
    keithley_close2 = 0;

    %keithley2 Table row length2
    global keithley_data_table_size2;
    keithley_data_table_size2 = 0;

    %Initialization for keithley2 graph
    Ilevel2=1E-6;  % Current in A
    Vlevel2=3;
    VlevelC2=num2str(Vlevel2);
    IlevelC2=num2str(Ilevel2);
    length2= 2000;
    fC2=num2str(update);
    gas2='EtOH';
    filename2= ['S1-' VlevelC2 'A-f' fC2 'Hz' '-gas2' gas2 '-2'];

%--------------------------------------------------------------------------


%--------------------------GUI CALLBACKS-----------------------------------
%COMM Port Dropdown
    function ports_dropdown_callback(hObject, eventdata, handles)
        val = get(hObject,'Value');
        string_list = get(hObject,'String');
        selected_string = string_list{val};
        port = serial(selected_string, 'BAUD', 9600);
        fopen(port);

        %Communication will hang if data is sent before the port is ready
```

```
            pause(3);
            port_flag = 1;
            console_history = {console_history; 'Opening COM5'};
            set(console_editbox, 'String', console_history);
        end

    %Input Temperature Field
        function temp_editbox_callback(hObject, eventdata, handles)
        end

    %Start Stop Button
        function start_stop_pushbox_callback(hObject, eventdata, handles)
            if(start == 0)
                start = 1;
                set(start_stop_pushbox, 'String', 'Stop');
            else
                start = 0;
                set(start_stop_pushbox, 'String', 'Start');
            end
        end

    %P Field
        function p_editbox_callback(hObject, eventdata, handles)
        end

    %I Field
        function i_editbox_callback(hObject, eventdata, handles)
        end

    %D Field
        function d_editbox_callback(hObject, eventdata, handles)
        end

    %Set PID Buttion
        function pid_set_pushbox_callback(hObject, eventdata, handles)
            %Get the P value set by user
            p_value = str2num(get(p_editbox, 'String'));

            console_history{end+1} = ['Sending P = ' num2str(p_value)];
            set(console_editbox, 'String', console_history);
            set(console_editbox, 'ListboxTop', console_size);
            console_size = console_size + 1;

            %Get the I value set by user
            i_value = str2num(get(i_editbox, 'String'));

            console_history{end+1} = ['Sending I = ' num2str(i_value)];
            set(console_editbox, 'String', console_history);
            set(console_editbox, 'ListboxTop', console_size);
            console_size = console_size + 1;

            %Get the D value set by user
            d_value = str2num(get(d_editbox, 'String'));

            console_history{end+1} = ['Sending D = ' num2str(d_value)];
            set(console_editbox, 'String', console_history);
            set(console_editbox, 'ListboxTop', console_size);
            console_size = console_size + 1;

            set_temp = str2num(get(temp_editbox, 'String'));

            %Save the temperature set
            set_temp_history = [set_temp_history set_temp];

            console_history{end+1} = ['Sending Temp = ' num2str(set_temp_history(end))];
            set(console_editbox, 'String', console_history);
            set(console_editbox, 'ListboxTop', console_size);
            console_size = console_size + 1;

            console_history{end+1} = '-';
            set(console_editbox, 'String', console_history);
            set(console_editbox, 'ListboxTop', console_size);
            console_size = console_size + 1;
        end

        function exit_pushbox_callback(hObject, eventdata, handles)
            if(port_flag)
                fclose(port);
            end
            instrreset;
            exit = 1;
            close(window);
            close(window2);
            error('');
        end

        function console_editbox_callback(hObject, eventdata, handles)
        end

    %Keithley Pushbox
        function keithley_pushbox_callback(hObject, eventdata, handles)
            keithley_start = 1;

            %Don't run Keithley communication setup if done already
            if(~keithley)
                c=gpib('ics',0,24);%%%%%%%%%%%%%%%%Keithley source meter address

                fopen(c);
                fprintf(c,'reset()');
                % set(c, 'EOIMode', 'off');
                %ob=instrfind;
                %set(ob(1), 'EOSMode', 'write');

                set(c, 'EOSMode', 'write');

                %channel A apply current, measure voltage
                fprintf(c, 'smua.reset();');

                %channel B
                fprintf(c, 'smub.reset();');

                set(c, 'EOSMode', 'write');

                set(c, 'Timeout', 1 );

                %Channel A
                fprintf(c,'smua.source.func = smua.OUTPUT_DCVOLTS'); % Select current source function
                fprintf(c,'smua.source.autorangei = smua.AUTORANGE_ON'); %Set source range to 50mA
                fprintf(c,'display.smua.measure.func = display.MEASURE_OHMS'); %display Ohm on device
                fprintf(c,['smua.source.limitv = ',num2str(10,10)]);
                fprintf(c,['smua.source.limiti = 0',num2str(1E-3,10)]);
                %fprintf(c(1),'smua.measure.nplc=2;');  % Number of Power Line Cycles (NPLC), where 1 PLC for
                                            %60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).
                fprintf(c,'smua.sense= smua.SENSE_REMOTE'); %Enable 4-wire prob
                fprintf(c,'smua.measure.autorangev = smua.AUTORANGE_ON'); %set voltage range

                %Channel B
                fprintf(c,'smub.source.func = smub.OUTPUT_DCVOLTS'); % Select current source function
                fprintf(c,'smub.source.autorangei = smub.AUTORANGE_ON'); %Set source range to 50mA
                fprintf(c,'display.smub.measure.func = display.MEASURE_OHMS'); %display Ohm on device
                fprintf(c,['smub.source.limitv = ',num2str(10,10)]);
                fprintf(c,['smub.source.limiti = 0',num2str(1E-3,10)]);
                %fprintf(c(1),'smub.measure.nplc=2;');  % Number of Power Line Cycles (NPLC), where 1 PLC for
                                            %60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).
                fprintf(c,'smub.sense= smub.SENSE_REMOTE'); %Enable 4-wire prob
                fprintf(c,'smub.measure.autorangev = smub.AUTORANGE_ON'); %set voltage range

                fprintf(c,'beeper.enable = 1');


                %fprintf(c,['smua.source.leveli = ',num2str(Ilevel,10)]); %Set current leve with 10 precision
                fprintf(c,['smua.source.levelv = ',num2str(Vlevel,10)]); %Set current leve with 10 precision
                %fprintf(c,'smua.source.output = smua.OUTPUT_ON');

                %fprintf(c,['smub.source.leveli = ',num2str(Ilevel,10)]); %Set current leve with 10 precision
                fprintf(c,['smub.source.levelv = ',num2str(Vlevel,10)]); %Set current leve with 10 precision
                %fprintf(c,'smub.source.output = smub.OUTPUT_ON');
            end

            keithley=1;

            %If the Keithley window was closed, redraw all GUI elements
            if(keithley_close)

                window2 = figure('name', 'Keithley',...
                            'Units', 'characters',...
                            'Visible', 'off',...
                            'Position', [5 1 156 55]);

                keithley_graph_panel_A = uipanel('Title', 'Channel A',...
                                    'Units', 'characters',...
                                    'Position', [16 27 100 27],...
                                    'Parent', window2);

                keithley_graph_panel_B = uipanel('Title', 'Channel B',...
                                    'Units', 'characters',...
                                    'Position', [16 0 100 27],...
                                    'Parent', window2);

                keithley_ctrl_panel = uipanel('Title', 'Keithley Control',...
                                    'Units', 'characters',...
                                    'Position', [0 24.5 16 6.5],...
                                    'Parent', window2);

                keithley_data_panel = uipanel('Title', 'Data',...
                                    'Units', 'characters',...
                                    'Position', [116 0 40 27],...
                                    'Parent', window2);

                keithley_close_pushbox = uicontrol('Parent', keithley_ctrl_panel,...
                                        'Style', 'pushbutton',...
                                        'Units', 'characters',...
                                        'FontSize', fontsize,...
                                        'String', 'Close',...
                                        'Position', [1, 1, 13, 2],...
                                        'Callback', {@keithley_close_pushbox_callback});

                keithley_save_data_pushbox = uicontrol('Parent', keithley_ctrl_panel,...
                                        'Style', 'pushbutton',...
                                        'Units', 'characters',...
                                        'FontSize', fontsize,...
                                        'String', 'Save data',...
                                        'Position', [1, 3.5, 13, 1.5],...
                                        'Callback', {@keithley_save_data_pushbox_callback});
```

```
            keithley_close = 0;
        end

        figure(window2);

        axes('parent', keithley_graph_panel_A);
        subplot(3,1,1); ylabel('Voltage [V]');
        subplot(3,1,2); ylabel('Current [A]');
        subplot(3,1,3); ylabel('Resistance [\Omega]');

        axes('parent', keithley_graph_panel_B);
        subplot(3,1,1); ylabel('Voltage [V]');
        subplot(3,1,2); ylabel('Current [A]');
        subplot(3,1,3); ylabel('Resistance [\Omega]');
    end


%keithley2 Pushbox
    function keithley_pushbox_callback2(hObject, eventdata, handles)
        keithley_start2 = 1;

        %Don't run keithley2 communication setup if done already
        if(~keithley2)
            c2=gpib('ics',0,24);%%%%%%%%%%%%%%%%keithley2 source meter address

            fopen(c2);
            fprintf(c2,'reset()');
            % set(c2, 'EOIMode', 'off');
            %ob=instrfind;
            %set(ob(1), 'EOSMode', 'write');

            set(c2, 'EOSMode', 'write');

            %channel A apply current, measure voltage
            fprintf(c2, 'smua.reset();');

            %channel B
            fprintf(c2, 'smub.reset();');

            set(c2, 'EOSMode', 'write');

            set(c2, 'Timeout', 1 );

            %Channel A
            fprintf(c2,'smua.source.func = smua.OUTPUT_DCVOLTS'); % Select current source function
            fprintf(c2,'smua.source.autorangei = smua.AUTORANGE_ON'); %Set source range to 50mA
            fprintf(c2,'display.smua.measure.func = display.MEASURE_OHMS'); %display Ohm on device
            fprintf(c2,['smua.source.limitv = ',num2str(10,10)]);
            fprintf(c2,['smua.source.limiti = 0',num2str(1E-3,10)]);
            %fprintf(c2(1),'smua.measure.nplc=2;');  % Number of Power Line Cycles (NPLC), where 1 PLC for
                                        %60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).
            fprintf(c2,'smua.sense= smua.SENSE_REMOTE'); %Enable 4-wire prob
            fprintf(c2,'smua.measure.autorangev = smua.AUTORANGE_ON'); %set voltage range

            %Channel B
            fprintf(c2,'smub.source.func = smub.OUTPUT_DCVOLTS'); % Select current source function
            fprintf(c2,'smub.source.autorangei = smub.AUTORANGE_ON'); %Set source range to 50mA
            fprintf(c2,'display.smub.measure.func = display.MEASURE_OHMS'); %display Ohm on device
            fprintf(c2,['smub.source.limitv = ',num2str(10,10)]);
            fprintf(c2,['smub.source.limiti = 0',num2str(1E-3,10)]);
            %fprintf(c2(1),'smub.measure.nplc=2;');  % Number of Power Line Cycles (NPLC), where 1 PLC for
                                        %60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).
            fprintf(c2,'smub.sense= smub.SENSE_REMOTE'); %Enable 4-wire prob
            fprintf(c2,'smub.measure.autorangev = smub.AUTORANGE_ON'); %set voltage range
    fprintf(c2,'beeper.enable = 1');


    %fprintf(c2,['smua.source.leveli = ',num2str(Ilevel2,10)]); %Set current leve with 10 precision
    fprintf(c2,['smua.source.levelv = ',num2str(Vlevel2,10)]); %Set current leve with 10 precision
    %fprintf(c2,'smua.source.output = smua.OUTPUT_ON');

    %fprintf(c2,['smub.source.leveli = ',num2str(Ilevel2,10)]); %Set current leve with 10 precision
    fprintf(c2,['smub.source.levelv = ',num2str(Vlevel2,10)]); %Set current leve with 10 precision
    %fprintf(c2,'smub.source.output = smub.OUTPUT_ON');
end

keithley2=1;

%If the keithley2 window was closed, redraw all GUI elements
if(keithley_close2)

    window3 = figure('name', 'keithley2',...
                'Units', 'characters',...
                'Visible', 'off',...
                'Position', [5 1 156 55]);

    keithley_graph_panel_A2 = uipanel('Title', 'Channel A',...
                        'Units', 'characters',...
                        'Position', [16 27 100 27],...
                        'Parent', window3);

    keithley_graph_panel_B2 = uipanel('Title', 'Channel B',...
                        'Units', 'characters',...
                        'Position', [16 0 100 27],...
                        'Parent', window3);

    keithley_ctrl_panel2 = uipanel('Title', 'keithley2 Control',...
                        'Units', 'characters',...
                        'Position', [0 24.5 16 6.5],...
                        'Parent', window3);

    keithley_data_panel2 = uipanel('Title', 'Data',...
                        'Units', 'characters',...
                        'Position', [116 0 40 27],...
                        'Parent', window3);

    keithley_close_pushbox2 = uicontrol('Parent', keithley_ctrl_panel2,...
                            'Style', 'pushbutton',...
                            'Units', 'characters',...
                            'FontSize', fontsize,...
                            'String', 'Close',...
                            'Position', [1, 1, 13, 2],...
                            'Callback', {@keithley_close_pushbox_callback2});

    keithley_save_data_pushbox2 = uicontrol('Parent', keithley_ctrl_panel2,...
                            'Style', 'pushbutton',...
                            'Units', 'characters',...
                            'FontSize', fontsize,...
                            'String', 'Save data',...
                            'Position', [1, 3.5, 13, 1.5],...
                            'Callback', {@keithley_save_data_pushbox_callback2});

    keithley_close2 = 0;
end

figure(window3);

axes('parent', keithley_graph_panel_A2);
subplot(3,1,1); ylabel('Voltage [V]');
```

```
        subplot(3,1,2); ylabel('Current [A]');
        subplot(3,1,3); ylabel('Resistance [\Omega]');

        axes('parent', keithley_graph_panel_B2);
        subplot(3,1,1); ylabel('Voltage [V]');
        subplot(3,1,2); ylabel('Current [A]');
        subplot(3,1,3); ylabel('Resistance [\Omega]');
    end


%Save Data Button
    function save_data_pushbox_callback(hObject, eventdata, handles)

       %Write to the console
        console_history{end+1} = ['Saved ' get(filename_editbox_label, 'String')];
       set(console_editbox, 'String', console_history);
       set(console_editbox, 'ListboxTop', console_size);
       console_size = console_size + 1;

       set(graph_panel, 'Units', 'pixels');

        %Define the area of the graph, and save a picture in Temp/Graphs
        %Save the data in Temp/Results
       %http://www.mathworks.com/matlabcentral/newsreader/view_thread/301782
       rect_pos = get(graph_panel, 'Position');
       f = getframe(window, rect_pos);
       [im, map] = frame2im(f);
       imwrite(im, ['Temp/Graphs/' get(filename_editbox_label, 'String') '.tiff']);
       save(['Temp/Results/' get(filename_editbox_label, 'String') '.mat'], 'meas_temp_history', 'pwm_history'   ); % saves
WS variable
       set(graph_panel, 'Units', 'Characters');
    end

%Keithley Save Data Button
    function keithley_save_data_pushbox_callback(hObject, eventdata, handles)
        set(keithley_graph_panel_A, 'Units', 'pixels');

        %Define the area of the graph, and save a picture in Temp/Graphs
        %Save the data in Temp/Results
       %http://www.mathworks.com/matlabcentral/newsreader/view_thread/301782
        rect_pos = get(keithley_graph_panel_A, 'Position');
        f = getframe(window2, rect_pos);
        [im, map] = frame2im(f);
        imwrite(im, ['Keithley/Graphs/' filename '.tiff']);
        save(['Keithley/Results/' filename '.mat'], 'rpos', 'vpos', 'ipos'); % saves WS variable
        %saveas(keithley_graph_panel_A,['Graphs/' filename '.tiff'], 'tiffn')  %saves in uncompressed tiff.
        %saveas(keithley_grap_panel,['Graphs/' filename '.fig'], 'fig')  %saves as matlab fig.
        set(keithley_graph_panel_A, 'Units', 'Characters');
    end


%keithley2 Save Data Button
    function keithley_save_data_pushbox_callback2(hObject, eventdata, handles)
        set(keithley_graph_panel_A2, 'Units', 'pixels');

        %Define the area of the graph, and save a picture in Temp/Graphs
        %Save the data in Temp/Results
       %http://www.mathworks.com/matlabcentral/newsreader/view_thread/301782
        rect_pos = get(keithley_graph_panel_A2, 'Position');
        f = getframe(window3, rect_pos);
        [im, map] = frame2im(f);
        imwrite(im, ['keithley2/Graphs/' filename2 '.tiff']);
        save(['keithley2/Results/' filename2 '.mat'], 'rpos', 'vpos', 'ipos'); % saves WS variable
        %saveas(keithley_graph_panel_A2,['Graphs/' filename2 '.tiff'], 'tiffn')  %saves in uncompressed tiff.
        %saveas(keithley_grap_panel,['Graphs/' filename2 '.fig'], 'fig')  %saves as matlab fig.
        set(keithley_graph_panel_A2, 'Units', 'Characters');
    end


%Keithley Close
    function keithley_close_pushbox_callback(hObject, eventdata, handles)
        keithley_start = 0;
        close(window2);
        keithley_close = 1;
    end


%keithley2 Close
    function keithley_close_pushbox_callback2(hObject, eventdata, handles)
        keithley_start2 = 0;
        close(window3);
        keithley_close2 = 1;
    end

%--------------------------------------------------------------------------


%--------------------------Main Loop---------------------------------------
    %Initialize variables
    time = 0;
    i = 1;
    meas_temp_history = 30  ;
    pwm_history = 0;
    debug1_history = 0;
    debug2_history = 0;
    debug3_history = 0;

    %Set up axes
    axes('parent', graph_panel);

    set(0, 'CurrentFigure', window);
    subplot(1, 2, 1);
    graph_1 = plot(time, meas_temp_history);
    set(gca, 'FontSize', fontsize);
    ylabel('Temperature (°C)', 'FontSize', fontsize);
    subplot(1, 2, 2);
    graph_2 = plot(time, pwm_history);
    set(gca, 'FontSize', fontsize);
    ylabel('PWM', 'FontSize', fontsize);

    while(1)
        set(graph_1, 'XData', time);
        set(graph_1, 'YData', meas_temp_history);
        set(graph_2, 'XData', time);
        set(graph_2, 'YData', pwm_history);

        refreshdata;
        drawnow;

        if(exit == 1)
            close all;
            return;
        end

        if(port_flag)
                                            %Send P, I, D to the Arduino
            fwrite(port, start);
            fwrite(port, p_value);
            fwrite(port, i_value);
            fwrite(port, d_value);
            %Write the last entry in temperature history
            fwrite(port, set_temp_history(end));

            %Read the available bytes in the buffer
            status = get(port, {'BytesAvailable'});
            %read_temp is a 1x1 cell array
            %Loop until measured temperature is available
            while ~status{1,1}
                status = get(port, {'BytesAvailable'});
                pause(0.1);
            end

            %Read the temperature and PWM
            meas_temp = fscanf(port, '%f');
            meas_temp_history = [meas_temp_history meas_temp];
            pwm = fscanf(port, '%f');
            pwm_history = [pwm_history pwm];

            time = [time time(end)+1];

            if(data_table_size ~= 1)
                clear mtable;
            end

           %There is no way to control the scrollbar in uitable. The cursor will move to the first row ever time a now row is
                %added.
           %To work around this, the old version of uitable returns a reference that is used to get the java reference of the
                %table.
           %With the java reference, manually set the cursor selection to be the last row.
           %Note that 'v0' in uitable ports the old version of uitable to the latest version.
            %http://www.mathworks.com/matlabcentral/newsreader/view_thread/165066
            mtable = uitable('v0', window, [rot90(meas_temp_history, 3) rot90(pwm_history, 3)], {'Temp', 'PWM'});
            jtable = mtable.getTable;
            jtable.setRowSelectionAllowed(0);
            jtable.setColumnSelectionAllowed(0);
            jtable.changeSelection(data_table_size,2-1, false, false);
            set(mtable.UIContainer, 'Position', [data_table_pos_x data_table_pos_y data_table_pos_w data_table_pos_h]);

            data_table_size = data_table_size + 1;

        end

        %Keithley
        if(keithley)
            if(keithley_start)
                set(0, 'CurrentFigure', window2);
                fprintf(c,['smua.source.levelv = ',num2str(Vlevel,10)]);
                fprintf(c,['smub.source.levelv = ',num2str(Vlevel,10)]);

                %fprintf(c,['smua.source.leveli = ',num2str(Ilevel,10)]);

                fprintf(c,'smua.source.output = smua.OUTPUT_ON');
                fprintf(c,'smub.source.output = smub.OUTPUT_ON');

                fprintf(c,'print(smua.measure.i())'); %reads voltage
                fprintf(c,'print(smua.measure.v())'); %reads current
                fprintf(c,'print(smua.measure.r())'); %reads resistance

                fprintf(c,'print(smub.measure.i())'); %reads voltage
                fprintf(c,'print(smub.measure.v())'); %reads current
                fprintf(c,'print(smub.measure.r())'); %reads resistance

                axes('parent', keithley_graph_panel_A);
```

```
            i1=fscanf(c); ipos_A(n)=str2double(i1);
            subplot(3,1,1);
            plot([1:1:n],ipos_A, 'Marker', 'x');ylabel('Current [A]');

            v1=fscanf(c); vpos_A(n)=str2double(v1);
            subplot(3,1,2);
            plot([1:1:n],vpos_A, 'Marker', 'x');ylabel('Voltage [V]');

            r1=fscanf(c); rpos_A(n)=str2double(r1);
            subplot(3,1,3);
            plot([1:1:n],rpos_A, 'Marker', 'x');ylabel('Resistance [\Omega]');

            axes('parent', keithley_graph_panel_B);

            i1=fscanf(c); ipos_B(n)=str2double(i1);
            subplot(3,1,1);
            plot([1:1:n],ipos_B, 'Marker', 'x');ylabel('Current [A]');

            v1=fscanf(c); vpos_B(n)=str2double(v1);
            subplot(3,1,2);
            plot([1:1:n],vpos_B, 'Marker', 'x');ylabel('Voltage [V]');

            r1=fscanf(c); rpos_B(n)=str2double(r1);
            subplot(3,1,3);
            plot([1:1:n],rpos_B, 'Marker', 'x');ylabel('Resistance [\Omega]');


            fprintf(c,'smua.source.output = smua.OUTPUT_OFF');
            fprintf(c,'smub.source.output = smub.OUTPUT_OFF');
            pause(1/update) %timeout to get stabel current and voltage
            Ilevel=Ilevel;
            %fprintf(c,'beeper.beep(0.1, 453)');
            %


           %There is no way to control the scrollbar in uitable. The cursor will move to the first row ever time a now row is
            %added.
           %To work around this, the old version of uitable returns a reference that is used to get the java reference of the
            %table.
           %With the java reference, manually set the cursor selection to be the last row.
           %Note that 'v0' in uitable ports the old version of uitable to the latest version.

            if(keithley_start)
            keithley_mtable_A = uitable('v0', window2, [rot90(ipos_A,3) rot90(vpos_A,3) rot90(rpos_A,3)], {'Current',
'Voltage', 'Resistance'});
            keithley_jtable_A = keithley_mtable_A.getTable;
            keithley_jtable_A.setRowSelectionAllowed(0);
            keithley_jtable_A.setColumnSelectionAllowed(0);
            keithley_jtable_A.changeSelection(keithley_data_table_size,3-1, false, false);
            set(keithley_mtable_A.UIContainer, 'Position', [keithley_data_table_pos_x keithley_data_table_pos_y
keithley_data_table_pos_w keithley_data_table_pos_h]);

            keithley_data_table_size = keithley_data_table_size + 1;
            end

            if(keithley_start)
            keithley_mtable_B = uitable('v0', window2, [rot90(ipos_B,3) rot90(vpos_B,3) rot90(rpos_B,3)], {'Current',
'Voltage', 'Resistance'});
            keithley_jtable_B = keithley_mtable_B.getTable;
            keithley_jtable_B.setRowSelectionAllowed(0);
            keithley_jtable_B.setColumnSelectionAllowed(0);
            keithley_jtable_B.changeSelection(keithley_data_table_size,3-1, false, false);
            set(keithley_mtable_B.UIContainer, 'Position', [keithley_data_table_pos_x 0 keithley_data_table_pos_w
keithley_data_table_pos_h]);
```

```
        keithley_data_table_size = keithley_data_table_size + 1;
        end

        n=n+1;
    end
end


%keithley2
if(keithley2)
    if(keithley_start2)
        set(0, 'CurrentFigure', window3);
        fprintf(c2,['smua.source.levelv = ',num2str(Vlevel2,10)]);
        fprintf(c2,['smub.source.levelv = ',num2str(Vlevel2,10)]);

        %fprintf(c2,['smua.source.leveli = ',num2str(Ilevel2,10)]);

        fprintf(c2,'smua.source.output = smua.OUTPUT_ON');
        fprintf(c2,'smub.source.output = smub.OUTPUT_ON');

        fprintf(c2,'print(smua.measure.i())'); %reads voltage
        fprintf(c2,'print(smua.measure.v())'); %reads current
        fprintf(c2,'print(smua.measure.r())'); %reads resistance

        fprintf(c2,'print(smub.measure.i())'); %reads voltage
        fprintf(c2,'print(smub.measure.v())'); %reads current
        fprintf(c2,'print(smub.measure.r())'); %reads resistance

        axes('parent', keithley_graph_panel_A2);

        i1=fscanf(c2); ipos_A2(n2)=str2double(i1);
        subplot(3,1,1);
        plot([1:1:n2],ipos_A2, 'Marker', 'x');ylabel('Current [A]');

        v1=fscanf(c2); vpos_A2(n2)=str2double(v1);
        subplot(3,1,2);
        plot([1:1:n2],vpos_A2, 'Marker', 'x');ylabel('Voltage [V]');

        r1=fscanf(c2); rpos_A2(n2)=str2double(r1);
        subplot(3,1,3);
        plot([1:1:n2],rpos_A2, 'Marker', 'x');ylabel('Resistance [\Omega]');

        axes('parent', keithley_graph_panel_B2);

        i1=fscanf(c2); ipos_B(n2)=str2double(i1);
        subplot(3,1,1);
        plot([1:1:n2],ipos_B, 'Marker', 'x');ylabel('Current [A]');

        v1=fscanf(c2); vpos_B(n2)=str2double(v1);
        subplot(3,1,2);
        plot([1:1:n2],vpos_B, 'Marker', 'x');ylabel('Voltage [V]');

        r1=fscanf(c2); rpos_B(n2)=str2double(r1);
        subplot(3,1,3);
        plot([1:1:n2],rpos_B, 'Marker', 'x');ylabel('Resistance [\Omega]');


        fprintf(c2,'smua.source.output = smua.OUTPUT_OFF');
        fprintf(c2,'smub.source.output = smub.OUTPUT_OFF');
        pause(1/update) %timeout to get stabel current and voltage
        Ilevel2=Ilevel2;
        %fprintf(c2,'beeper.beep(0.1, 453)');
        %
```

```
            %There is no way to control the scrollbar in uitable. The cursor will move to the first row ever time a now row is
             %added.
            %To work around this, the old version of uitable returns a reference that is used to get the java reference of the
             %table.
            %With the java reference, manually set the cursor selection to be the last row.
            %Note that 'v0' in uitable ports the old version of uitable to the latest version.

            if(keithley_start2)
            keithley_mtable_A2 = uitable('v0', window3, [rot90(ipos_A2,3) rot90(vpos_A2,3) rot90(rpos_A2,3)], {'Current',
'Voltage', 'Resistance'});
            keithley_jtable_A2 = keithley_mtable_A2.getTable;
            keithley_jtable_A2.setRowSelectionAllowed(0);
            keithley_jtable_A2.setColumnSelectionAllowed(0);
            keithley_jtable_A2.changeSelection(keithley_data_table_size2,3-1, false, false);
            set(keithley_mtable_A2.UIContainer, 'Position', [keithley_data_table_pos_x2 keithley_data_table_pos_y2
keithley_data_table_pos_w2 keithley_data_table_pos_h2]);

            keithley_data_table_size2 = keithley_data_table_size2 + 1;
            end

            if(keithley_start2)
            keithley_mtable_B2 = uitable('v0', window3, [rot90(ipos_B,3) rot90(vpos_B,3) rot90(rpos_B,3)], {'Current',
'Voltage', 'Resistance'});
            keithley_jtable_B2 = keithley_mtable_B2.getTable;
            keithley_jtable_B2.setRowSelectionAllowed(0);
            keithley_jtable_B2.setColumnSelectionAllowed(0);
            keithley_jtable_B2.changeSelection(keithley_data_table_size2,3-1, false, false);
            set(keithley_mtable_B2.UIContainer, 'Position', [keithley_data_table_pos_x2 0 keithley_data_table_pos_w2
keithley_data_table_pos_h2]);

            keithley_data_table_size2 = keithley_data_table_size2 + 1;
            end

            n2=n2+1;
          end
        end


        pause(update)
    end
%--------------------------------------------------------------------------

end
```

APPENDIX D – Arduino Code

```
#include <math.h>

#define ERRORWINDOW 20

float p_value = 0; //User P for testing right now
float i_value = 0; //User I
float d_value = 0; //User D
float set_temp = 0; //User set temperature

float calc_temp = 0; //Calculated temperature

float v_o = 0; //Measured voltage
float v_i = 0; //Calculated diff amp input voltage
float v_j = 0; //Junction voltage
float v_ss = 5; //Whestone bridge reference voltage
float gain = 1.4; //Diff amplifier gain adjustable
float v_therm = 0;
float v_adj = 1.6; //

float R1 = 10000; //R1 in Ohm
float R2 = 345; //R adjustable
float R3 = 10000; //R1 in Ohm
float R4 = 0; //R Thermister
float R25 = 10000; //Thermister at 25 celcius, from datasheet


float pwmHeat = 6; //PWM Heat pin
float pwmCool = 5; //PWM Cool pin
float ch1_p = 1;
float ch1_n = 0;

float v_o_n = 0;
float v_o_p = 0;
//PID variables
float last_calc_temp = 0;
float error = 0;
float last_error = 0;

float p_error = 0;
float i_error = 0;
float d_error = 0;
float pid_sum = 0; //Sum of P, I, and D elements

float error_sum = 0;
float errors[ERRORWINDOW];

int t_d = 500; //Loop delay in miliseconds

int state = 0;
int fill_error = 0;

float start = 0;

void setup() {
		Serial.begin(9600);  // opens serial port, sets data rate to 9600 bps
}

void loop() {
		ReadFromSerial();
		ReadFromAnalog();
		CalculateTemperature();

    if(start)
    {
```

```
                PID();
        }
        else
        {
                error = 0;
                error_sum = 0;
                pid_sum = 0;
                analogWrite(pwmHeat, 255);
                analogWrite(pwmCool, 255);
        }

        WriteToSerial();

        //Loop every t_d;
        delay(t_d);
}

int ReadFromSerial()
{
  //Read user inputs
  while(!Serial.available())
  {
  }
  start = Serial.read();

  while(!Serial.available())
  {
  }
        //Read P
  p_value = Serial.read();

  while(!Serial.available())
  {
  }
        //Read I
  i_value = Serial.read();

  while(!Serial.available())
  {
  }
        //Read D
  d_value = Serial.read();

  while(!Serial.available())
  {
  }
        //Read Set Temperature
  set_temp = Serial.read();

        return 0;
}

int ReadFromAnalog()
{
  //Measure Voltage
  v_o_n = analogRead(ch1_n);
  v_o_p = analogRead(ch1_p);

        return 0;
}

int CalculateTemperature()
{
        v_o = (v_o_p - v_o_n)*5/1024; //Convert to Vo
  v_i = v_o/gain; //Convert Vo to Vi- diff amplifier
```

```
            //Calculate RTD resistance in kOhm
            //R4 = (R3*(v_i/v_ss)+(R2*R3)/(R1+R2))/((1-R2/(R1+R2)-v_i/v_ss));
    v_therm = v_adj - v_i;
    R4 = v_therm*R3/(5-v_therm);
    //Convert to temperature
            //This equation was found by plotting the resistance to temperature chart
            //Replace this with the Steinhart-Hart equation
    calc_temp = -21.42*log(R4/R25)+31.069;
            return 0;
}

int PID()
{
    //PID compute PWM
    last_error = error;
    error = set_temp - calc_temp;

            CalculateP();
            CalculateI();
            CalculateD();

    pid_sum = p_error + i_error + d_error;

            LimitOutput();

  return 0;
}

int CalculateP()
{
            //Compute P term
    p_error = error*p_value;

            return 0;
}

int CalculateI()
{
            //If error array isn't full, add the new error
            if(fill_error < ERRORWINDOW)
            {
                    errors[fill_error] = error;
                    error_sum = error_sum + errors[fill_error];
                    fill_error = fill_error + 1;
            }
            //If the error array is full, shift all elements left by 1 and add the new error
            else
            {
                    error_sum = 0;
                    for(int j = 0; j < ERRORWINDOW-1; j++)
                    {
                            errors[j] = errors[j+1];
                            error_sum = error_sum + errors[j];
                    }
                    errors[ERRORWINDOW-1] = error;
                    error_sum = error_sum + errors[ERRORWINDOW-1];
            }
    //Compute I term
    i_error = error_sum*i_value*t_d/1000;

            return 0;
}

int CalculateD()
```

```
{
  //Compute D term
  d_error = (error-last_error)*d_value/t_d;

        return 0;
}

int WriteToSerial()
{
  //Send calculated temperature to user
  //Serial.println(calc_temp, 4);
  Serial.println(calc_temp, DEC);
        //Send calculated PWM to user
  Serial.println(pid_sum, 4);
        return 0;
}

int LimitOutput()
{
  if(pid_sum > 0)
  {
   //If PID is positive, send to pwmHeat
   if(pid_sum >= 255)
   {
                //Alias pid output if exceeds Arduino 100% PWM
                pid_sum = 254;
   }
   analogWrite(pwmHeat, 255-pid_sum);
   analogWrite(pwmCool, 255);
  }
  else
  {
   if(pid_sum < 0)
   {
    //If PID is negative, send to pwmCool
                if(pid_sum < -255)
                {
                 //Alias pid output if exceeds Arduino 100% PWM
                 pid_sum = -254;
                }
        analogWrite(pwmCool, 255+pid_sum);
    analogWrite(pwmHeat, 255);
   }
  }
 return 0;
}
```