TEMPERATURE CHAMBER CONTROL SYSTEM

BY REUBEN HUGHES AND ALEX RUDDELL

Contents

1. Introduction....3
2. Data Acquisition....3
3. Filtering....3
4. Voltage-to-temperature Conversion....3
5. Control System....4
6. Graphical User Interface....4
6.1 Manual and Automatic Mode Sections....4
6.2 Real Time Data Feed Section....4
6.3 System Settings Section....5
7. Task Priority....5
7.1 The Main Thread....5
7.2 Background Worker 1....5
7.3 Background Worker 2....5
8. Controller Performance....5
9. Conclusion....6
10. Appendices....6
10.1 Graph for 2-degree rise from Room Temperature....6
10. 2 Graph for 5-degree rise from Room Temperature....6

1. Introduction

The objective of this project is to produce a program that allows the user to control a temperature chamber. The role of the control system is to maintain the chamber at a constant temperature 2 to 5 degrees centigrade above room temperature. The temperature chamber employs a heating element, a cooling fan, and three thermistors to sense the resulting temperature inside the chamber. Voltage readings are to be taken from each thermistor at a rate of 10Hz or higher, and denoised using a digital FIR filter. Once denoised, they are converted to a temperature in degrees centigrade. The user interface must give the technical user complete control over the system. The user must be able to view each temperature reading from the thermistors, enable and disable these thermistors, and control the system through toggling between manual and automatic operation mode. It is important that the user interface has high priority so that does not freeze or lag due to other background processes.

2. Data Acquisition

To communicate with each thermistor, three instances of the class “AnalogI” were created. These objects are employed to read the voltage across each respective thermistor. Voltage data points are stored in a vector by the AnalogI object each time the ReadWaveForm() function is called. The sampling frequency and number of samples can be specified when the channel is opened, however the vector in which the data is stored is not in a usable form for a FIR filter application. To gather the discrete voltage values in a form suitable for applying the filter, the channel is made to take the minimum number of samples; 2. Then, the most recent value in the vector is copied to an array which is indexed as a ring buffer. The length of this ring buffer array is defined by the filter length. To meet the brief specification, the thermistor voltage values are read at a rate of 10Hz, however this can be easily changed to any desired sampling frequency. The ring buffer format is integral to the success of the correct operation of the filter; explained in the next section.

3. Filtering

The filtering method aims to filter the voltage before converting to a temperature value. This is both simple and accurate. As mentioned in the previous section, it was assumed that using the ReadWaveform() function within the AnalogI class would count as sufficient filtering. This functionality was edited to allow the user to pass in a filter window length and return an average of the values within that window. There were two main problems with this filter design. The first was that the window of samples was taken at an extremely high rate, so the values weren’t really being “smoothed” between each 0.1 second interval. The second problem was that the user didn’t have the ability to edit filter weights; the filter was simply an “n-point average” filter.

A separate filter function was created to solve these main issues. The user passes in filter weights through the “input.txt” filter file, and the window length is inherently determined by how many filter coefficients they create. The resulting filter incorporates a ring buffer of user-determined length that averages the values based on the given weights. Convolution is used to implement this filter; with the weight array reversed and applied at the current ring index. A unique feature of the filter is that the fraction weights don’t have to add up to a value of 1. This reduces the need for specific user input in the text file. This filter design allows for the user to customise how much of an effect past values have on the current output, weighting as they see fit.

4. Voltage-to-temperature Conversion

Once the voltage readings from the thermistors are filtered, they are converted to a temperature in degrees using the below thermistor equation that was provided in official lab documents.

$$T = \frac{T_0 \cdot B}{T_0 \cdot \ln(R/R_0) + B}$$

The values of B and R_0 are unique to each thermistor and were retrieved from data sheets on the RS-online website. The resistance of the thermistor, which changes with temperature, is found using the voltage divider formula relative to the 5V source. The output of the thermistor formula is in degrees kelvin, so is converted to degrees centigrade for display to the user. Once in the correct format, the mean is calculated and fed back to the control system as the current chamber temperature value.

The issue with using a mean value to describe the chamber temperature is, if a thermistor reading was extremely off or the device faulty, the recorded temperature would shift the mean by an extreme amount. For this reason, the user of the program has the option to exclude specific thermistors from this mean. The user

interface presents the raw temperature values received from each thermistor, and it is assumed that the user would be able to notice if a faulty thermistor situation arose, disabling it accordingly.

5. Control System

Implementing the controller required the identification of what control systems *could* be used. The fan and heater both only have two states; on or off, so a proportional controller could not be easily used. Therefore, an on/off controller was deemed most appropriate and implemented. The only control parameters are the two hysteresis boundaries and the target temperature for the system measure against. Below the lower hysteresis boundary, the fan is switched off and the heater switched on. This heats up the chamber and brings it closer to the target temperature. Above the upper hysteresis boundary, the fan is switched on and the heater switched off. This cools the chamber down. When the measured temperature is between the two boundaries, both the fan and heater are switched off. The ideal hysteresis boundaries would keep the temperature chamber within ±0.25 degrees centigrade of the target temperature, whilst also minimising frequent switching of the actuators. It was found from running multiple tests that the ideal lower hysteresis bound is approximately 0.1 degrees below the target temperature, and the upper hysteresis bound is equal to the target temperature. When the program is opened, the hysteresis bounds are set to these base “ideal” values. The reason why the hysteresis is asymmetrical about the target temperature is due to the heater having a delayed effect on the chamber compared to the fan, so the internal temperature still rises once its turned off.

6. Graphical User Interface (GUI)

As this GUI is for an analytical data program, the design approach was to create a simplistic interface that gives the user complete control over the system with ease. The GUI, shown to the right, is divided into three different sections to group all similar information together for an intuitive layout.

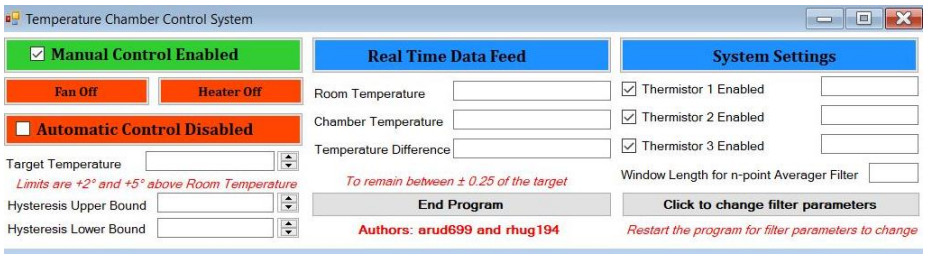


Figure 1: Graphical User Interface Design

6.1 Manual and Automatic Mode Sections

The left-most sections, “Manual Control” and “Automatic Control”, contain all information related to how the system is controlled.

Fan and heater toggle buttons are located under the “Manual” header and can only be used when in manual mode. Although they cannot be used to operate the machine when in automatic mode, they still change appearance based on the control action that the program takes, so that the user is aware of the heater and fan states. These buttons display a green or red colour to accompany the “on” or “off” text changes.

The “Automatic” header encases buttons for changing the control process variables. The target temperature, upper hysteresis bound, and lower hysteresis bound can all be changed via respective “up-down” buttons and are limited to set ranges. The range of target temperatures, as specified in the brief, is between 2 and 5 degrees centigrade above room temperature. The button iterates in full degrees. Room temperature is initialised as the system temperature on program launch. The hysteresis bounds are each restricted between 0 and 1 degree(s) above and below the target temperature. These buttons adjust the bounds in 0.05-degree increments.

6.2 Real Time Data Feed Section

The centre section presents data to the user in “real time”, with dynamic variables updating every half-second. It displays the reference room temperature, and the dynamic variables; chamber temperature and temperature difference. The middle section also contains an “End Program” button that, when pressed, cools the chamber back to approximately room temperature before closing the form. This same functionality can be achieved through pressing the “Close” button in the top right corner, standard for all Windows programs. This section aims to easily present information to the user such that they can decide what action to take - whether the control be automatic or manual in nature.

6.3 System Settings Section

The right-most section, “System Settings”, contains user-changeable variables relating to the operation of the system itself, rather than the control process. In this section, the user can enable / disable thermistors based on their displayed temperature. In this way, if one of the thermistors is faulty or displaying an extremely different value to the others, it can be disabled, therefore ceasing its contribution to the control process. The filter length parameter is also displayed alongside a button linking the “input.txt” filter file. It is expected that the user will restart the program on changing the filter weights, as they are only read once at the start of the program. This design choice was made to reduce the probability of system errors occurring, as attempting to edit a .txt file when it is being read by a program could result in issues.

7. Task Priority

Because this program runs from the computer and does not have to be loaded to a small embedded system, there are a lot more resources available for achieving correct task priority. Our program contains three threads; the main thread, the “backgroundWorker1” thread, and the “backgroundWorker2” thread. These threads run simultaneously on, so are never blocking one another.

Multi-threading allows for further program abstraction. The software developer can worry less about how long each task takes in relation to one another as different threads use different hardware resources. In this way, the final program produced runs the control system exclusively on one thread, one thread is responsible for calculations, and the other handles all non-vital program operations which do not need specific priority. Although the program could be operated on a single thread, segmenting into multiple threads improves system response time and efficiency.

7.1 The Main Thread

The main thread contains all system variable initialisation, function declaration, and form instantiation. As such, most of its processing is done at the launch of the program, which doesn’t contribute to periodic processing in this thread. Most of the processes that run at steady state are the event handlers responding to form interactions by the user. This includes changes in checkbox states, button presses, and timer ticks. These are technically “interrupts”, so are kept separate from calculation and control system threads. The periodic operation stems from timer ticks. Two timers; timer 1 and timer 2, are present in the main thread. At 0.5-second intervals, timer 1 updates the state of dynamic variables on the user interface. At 0.1-second intervals, timer 2 appends specified data to a text file, and updates the appearance of buttons that need a quicker response time, such as the fan and heater toggle buttons. The file writing functionality is performed in this thread as it must be synchronised with the GUI output. Running all input / output functionality in a separate thread was considered, but for the current system response time not deemed necessary. There is no observable lag as-is, so attempting to synchronise two different threads wouldn’t be worth the extra effort.

7.2 Background Worker 1

This thread is dedicated to reading values from the AnalogI input channels and calculating the average temperature between thermistors at a frequency of 10 Hz. Before the continuous loop is entered, this process also initialises most display text boxes with initial-state values. This thread aims to contain most, if not all, of the processor-heavy operations of the system.

7.3 Background Worker 2

This thread takes care of setting the initial states of the fan and heater as “off” before entering the continuous loop. This continuous loop is responsible for the control of the system. The infinite loop continuously checks if automatic mode has been enabled or not. When enabled, this loop calls the control system function. How this control system works is explained earlier in this report under Section 5. Although this operation doesn’t necessarily have processor-heavy requirements, accurate timing and running of the system is important for calculation accuracy and safety.

8. Controller Performance

The controller that was implemented successfully maintained the chamber within ±0.25 of the temperature specified by the user. Rise time of the control system is minimised as, being an on/off control system, the control effort is at a maximum while approaching the desired temperature. There is some overshoot in Appendix I, where the temperature of the chamber is above the 0.25-degree boundary. This overshoot lasts for around 20 seconds, compared to a rise time of about 70 seconds, so it was decided that the overshoot is

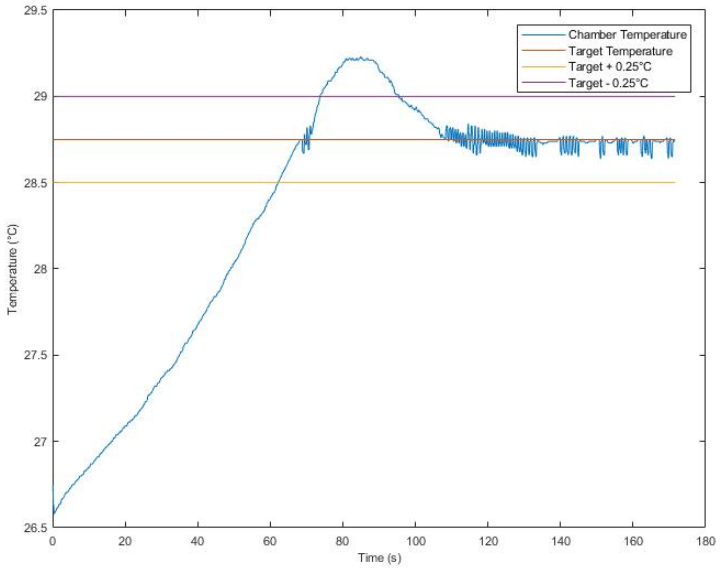
acceptable. Two plots of general system response can be found in the appendices. Appendix I shows the response for a 2-degree rise from room temperature, and Appendix II show the response for a 5-degree rise.

9. Conclusion

The final solution for this project meets all the requirements of the brief. The solution successfully implements a GUI that interacts with the sensors and actuators within the temperature chamber. The end user can communicate with a control system that can maintain the chamber at a defined temperature. Many settings can be modified by the user, including target temperature, hysteresis boundaries, and sensor activity. One assumption that must be satisfied for this GUI to work successfully is that the user has a reasonable understanding of the control system and situation, or “scope”. The reason for this is that the brief states the user needs the ability to change the control system parameters. The user is also given the option to disable sensors if a sensor is malfunctioning or the user wants to observe specific thermodynamic behaviour of the system. The functionality of the program was designed to have low coupling, meaning that features can be easily modified or added to the program without needing major changes to the rest of the program. Through satisfying all requirements in the brief, it is believed that the resulting system is a successful implementation of C# code and Windows Forms objects for this specific use. Ultimately, a system was created that was well received and performed well on assessment day. If our main end user, the tutor assessing our team, was pleased with the program in its entirety, then we consider the project a major success.

10. Appendices

Appendix I: Graph for 2-degree rise from Room Temperature



Appendix II: Graph for 5-degree rise from Room Temperature