Steam Heat Controller Retrofit

Final Report

Team - sddec18-02

Jevay Aggarwal - Technical Lead

Sarah Coffey - Weekly Report Lead

Thomas Devens - Project Plan Lead

Joseph Filbert - Client Contact Lead

Ken Wendt - Website Master

Liz Wickham Kolstad - Design Document Lead

Lee Harker - Client and Advisor

http://sddec18-02.sd.ece.iastate.edu/ - Website

sddec18-02@iastate.edu - Contact

Revised: December 3, 2018

Table of Contents

1. Introductory material 5
    1.1 Acknowledgement 5
    1.2 Definitions and terminology 5
    1.3 Problem statement 5
    1.4 Operating environment 5
    1.5 Intended users and uses 5
    1.6 Assumptions and limitations 6
    1.7 Expected end product and other deliverables 6
2. Project Design 7
    2.1 Requirements 7
    2.2 Considerations 7
    2.3 Overall Design 8
    2.4 MCU Design 8
        2.4.1 Motor and mount 8
        2.4.2 Hardware 9
        2.4.3 Software 9
    2.5 RCU Design 10
        2.5.1 Hardware 10
        2.5.2 Software 11
    2.6 WCU Design 11
        2.6.1 Hosting 11
        2.6.2 Database 11
        2.6.3 Functionality 11
        2.6.4 Errors 11
3. Implementation Details 12
    3.1 MCU 12
        3.1.1 Motor and mount 12
        3.1.2 Hardware 13
        3.1.3 Software 15
    3.2 Remote Control Unit (RCU) 15
        3.2.1 Hardware 15
        3.2.2 Software 16
    3.3 Web Control Unit (WCU) 16
        3.3.1 Hosting 16
        3.3.2 Database 17
        3.3.3 Functionality 17
        3.3.4 Errors 19

4. Testing Process and Results 20
4.1 Functional Testing and Results 20
4.1.1 MCU 20
4.1.2 RCU 21
4.1.3 Temperature Sensor Testing 21
4.2 Non-functional Testing and Results 22
4.2.1 Battery 22
4.2.2 Error notifications 23
5. Related Products 23
6. Final status 23
6.1 MCU 23
6.2 RCU 24
6.3 WCU 24
6.4 Integration Testing 24
6. Appendix I - Operations Manual 25
6.1 Setup 25
6.1.1 MCU 25
6.1.2 RCU 26
6.1.3 WCU 27
6.2 Operation 27
6.2.1 MCU 27
6.2.2 RCU 27
6.2.3 WCU 27
7. Appendix II - Alternative and Past Iterations of Design 28
7.1 Prior group 28
7.2 RCU 28
7.3 Motor Mount 29
8. Appendix III - Lessons Learned 30
8.1 Security 30
8.2 Temperature Sensor Placement 31

Table of Figures

Figure 1. 8
Figure 2. 9
Figure 3. 10
Figure 4. 12
Figure 5. 13
Figure 6. 14
Figure 7. 14
Figure 9. 15
Figure 10. 16
Figure 11. 18
Figure 12. 18
Figure 13. 19
Figure 14. 19
Figure 15. 20
Figure 16. 21
Figure 17. 22
Figure 18. 25
Figure 19. 26
Figure 20. 29
Figure 21. 30

1. Introductory material

1.1 Acknowledgement

Our client and faculty advisor, Leland Harker, will be our point of contact for any equipment we need as well as any fabrication of specific parts. Additionally, he will provide technical advice on various parts of the project, and guidance on the project.

1.2 Definitions and terminology

Term	Definition
MCU	Motor Control Unit
RCU	Remote Control Unit
WCU	Web Control Unit
Block	Set of rooms whose temperature is controlled by a single valve

1.3 Problem statement

As with many buildings on Iowa State University's campus, Coover Hall relies on steam to heat the building. Unfortunately, the room temperature is controlled by valves that are commonly hidden behind furniture or equipment making it difficult to adjust the temperature. Additionally, there is no reliable way to control the room temperature consistently as the room is subjected to outside weather and other factors and the valve cannot compensate for these variances. In other words, the only way to adjust the temperature is to open or close the valve manually and wait to see if the temperature changes desirably. There are also no mechanisms in place to control the valves remotely or en masse.

1.4 Operating environment

Our product will be located in the older parts of Coover Hall. The steam valves are in faculty offices, graduate student labs, and public meeting rooms. The product will be operating in rooms that are expected to stay within normal indoor temperature ranges (64-80 °F). The retrofit may or may not be visible or physically accessible, based on the valve location in the room. The remote controller will be in the same room as the retrofit. Some of the valves are located in rooms that will have dust and debris in the air, so the electrical equipment needs to be packaged properly.

1.5 Intended users and uses

The users will primarily be the faculty and staff with offices and labs that are heated with steam valves. Since a single steam valve controls the heat for several rooms, the thermostat will be placed in an easily accessible location so whoever is using the room or lab will be able to change the temperature. This is our base requirement that needs to be met. Ideally, we would like to create a network interface that ETG could use to monitor and manipulate each individual thermostat in Coover. This could be used to automatically decrease the temperature during breaks where fewer people will be using the building. This would save money and be a more efficient use of steam heating.

The design plan is to rely on the temperature sensor located on the MCU for collection of data. This being the actual temperature in the room and the desired temperature given by the user. Based on the difference between the current temperature and the desired temperature, the MCU will determine which way and how much to change the position of the valve. It will also monitor and detect any issues with the valve or database, sending error messages to be displayed on the thermostat, as well as sending an email to any staff that have remote access.

1.6 Assumptions and limitations

Assumptions

- The product will only be used in Coover Hall.
- It will be installed in one room with the capability to replicated in an unlimited amount of rooms.
- The temperature in Coover will be within 60 °F to 80 °F.
- There is a wall outlet near the steam valve to power the motor control unit.

Limitations

- Our product has a $500 budget for building and testing our whole project.
- The thermostat must be battery operated and be able to last for at least one semester.
- The valve controller must be fitted to the valve with little to no change to existing setup.
- The remote access must be secure i.e. only accessible to the staff of Coover Hall.

1.7 Expected end product and other deliverables

The remote control unit will handle the local input of temperature adjustments and display any error messages. It will get the information and error data from the MCU and submit temperature changes to the motor control unit.

The motor control unit will take the variables of current temperature and desired temperature. It will then perform a computation to determine the direction and rotation amount that the motor needs to turn the valve. The motor controller will maintain the desired temperature until a different temperature is input. The programming on the device will also check for errors with the motor, mitigate any issues found, and report them when human intervention is needed.

The web control unit will allow users to change the temperature via a web page. It will also provide the interface to resolve reported errors and set mass control of the valves. Error reports will also be generated through this unit.

2. Project Design

2.1 Requirements

Functional

- Retrofit will be motor-driven.
- Retrofit will be controlled wirelessly.
- There will be one retrofit, remote controller, and temperature sensor per valve.
- Motor and communication errors will be recorded and handled automatically.
- Ability to control all connected valves en masse.

Non-functional

- Temperature will be held within ± 1 °F.
- Remote controller must be powered by a battery with a lifespan of 1 semester.
- Errors will be reported timely (within 30 minutes).

2.2 Considerations

The users of the system are expected to interact with our product like they would with a normal thermostat. This means that no previous knowledge of or interaction with thermostats is necessary. Changing the temperature in the room should be as straightforward as pressing a button. Our design needs to reflect that simplicity and ease of use for the potential user.

There are a variety of steam valves in Coover Hall that would need to support this retrofit. Each valve is located in a different position in the room, with some being in the middle of a wall and others being located around other pipes and obstacles. It would be ideal to design the retrofit such that it could be fit on any of the valves in Coover Hall. Thus, we need to consider all possible installation environments.

The steam valves being retrofit will control the amount of heat in the room they are located in, as well as several adjacent rooms. As a result of having multiple rooms with temperatures controlled by one steam valve, each room may have a different temperature, and each valve will have several users. The design needs to consider how to handle multiple conflicting set temperature requests. Our client has determined that coordination of multiple room temperatures to control a specific valve is out of scope for this project.

The battery life on the remote control unit is also a considerable factor in the design. Since it needs to last for an entire semester, we need to find an energy efficient solution. The bottom line for reducing energy consumption was to find a design that limits the amount of processing on this device.

Security within our system is also a factor in the design. The components will be connected within the Iowa State University network. As such, we should take the proper precautions in our designs to not expose information. We also need to ensure that the connections and components are not vulnerable to malicious users.

The devices we create and the processes we use to develop them need to be as safe as possible. Some activities where we need to practice standard safety precautions are in soldering, measuring voltage and current, operating shop equipment, and handling power sources. For the end design, our product should reduce the possibility of safety malfunctions by packaging our devices, properly insulating exposed connections, and ensuring all power and ground connections are secure.

2.3 Overall Design

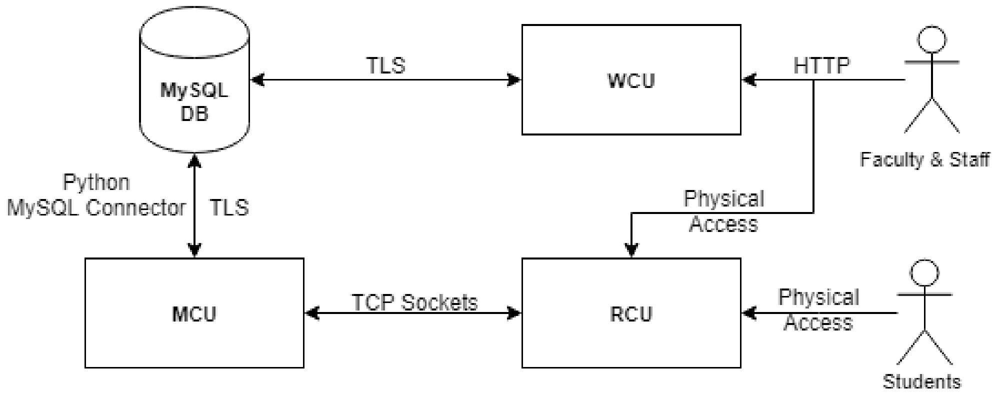


Figure 1. Block diagram for the system

2.4 MCU Design

2.4.1 Motor and mount

The retrofit for the valve needs to be capable of controlling the steam valve reliably. In order to accomplish this, we needed to use a motor with appropriate power and size. To determine the amount of torque needed by a motor, we used torque wrenches to measure how much would be needed on the valve; the result was approximately 8in-lb of torque to turn from a full stop and 1 in-lb of torque to move when not fully closed. We also wanted to find a motor that had a built-in encoder, so that we could monitor the motor's rotation in software, as well as minimize the amount of parts needed to put together. With a chosen motor, we could then design the mount around the valve and this motor. The optimal solution for the mount was to find a design that could be fit on any of the valves in Coover Hall. Some of the steam valves are located in corners with a grouping of other pipes, while others are located in the middle of open walls. We decided to design the mount to meet the physical space allowed by the worst positioned valve.

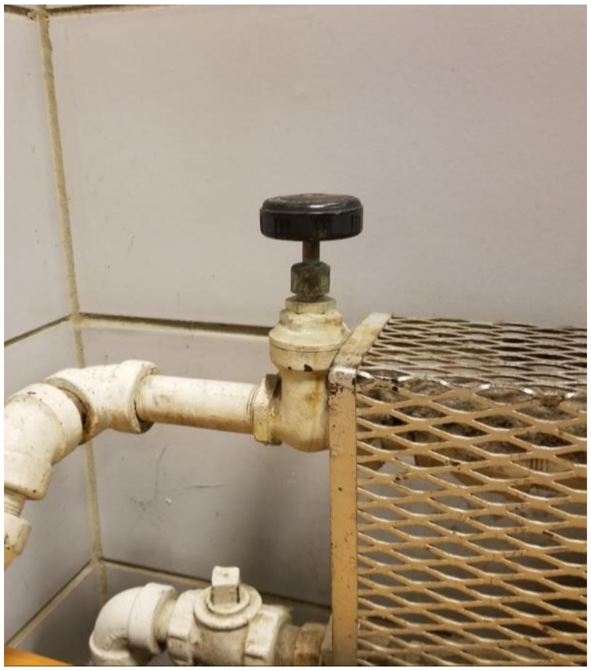


Figure 2. Valve location example

2.4.2 Hardware

We also needed to find a microcontroller that was capable of driving the motor in software. The microcontroller would need the appropriate amount and type of pins, the ability to connect to the network and to the RCU, and have sufficient memory and processing capability to run the software. Through several iterations of design, it was determined that the microcontroller would communicate with the RCU through TCP sockets over WiFi. The microcontroller would also need the ability to query the SQL database that was used to store the current temperature information and error codes. The temperature sensor was moved onto the MCU due to constraints on the RCU, so the microcontroller would also need to have the capability to interface with the chosen temperature sensor. Since these programs have the potential to be run at the same time, the microcontroller would need to support concurrent processes (ie, multithreading).

2.4.3 Software

Since the temperature for a single valve can be controlled by several users and may result in multiple, conflicting desired set temperatures, we decided that the MCU would only use the most recent temperature requested. Controlling the motor in software requires the ability check the current and set temperature and determine the amount of rotation necessary. Since the temperature will be fluctuating within the rooms, it was determined that a simple proportional system would be sufficient for this project.Thus, this software only needs to change the valve position relative to the difference between the set and current temperatures.

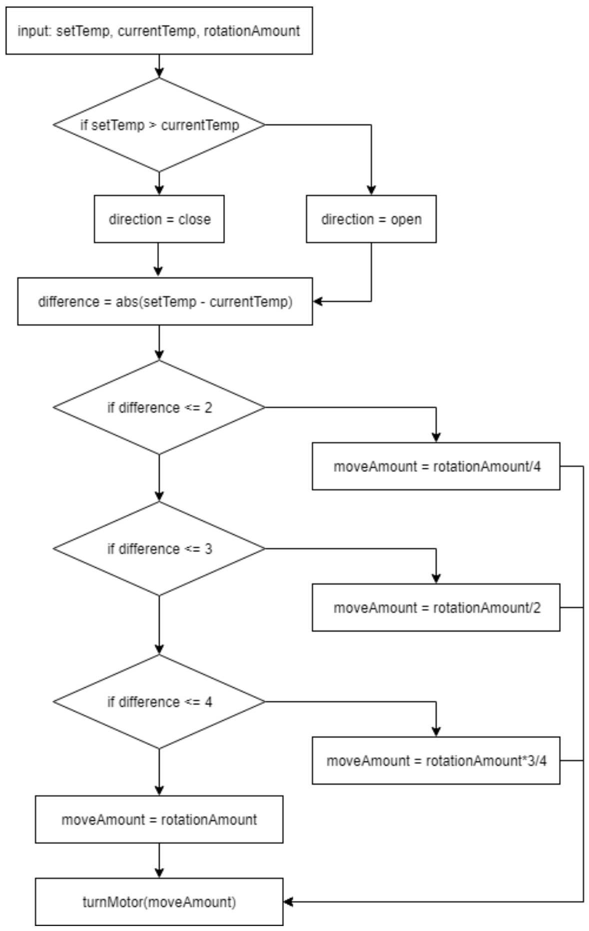

input: setTemp, currentTemp, rotationAmount
if setTemp > currentTemp
direction = close
direction = open
difference = abs(setTemp - currentTemp)
if difference <= 2
moveAmount = rotationAmount/4
if difference <= 3
moveAmount = rotationAmount/2
if difference <= 4
moveAmount = rotationAmount*3/4
moveAmount = rotationAmount
turnMotor(moveAmount)

Figure 3. Motor controller logic

The software to communicate with the RCU was decided to act as the server in the TCP Client-Server scheme, since the microcontroller on the RCU would always be powered on. The program would need to be able to wait for incoming connection requests from the client, get the current and set temperature to send to the RCU, send it, and wait for a new set temperature response from the client. With the decision to use a database as a central repository for this data, the program would also need to push the new temperature information to the database. Finally, there needed to be a program that routinely measured the current temperature and pushed it to the database for the WCU. Based on earlier design decisions, this program could be set to run periodically, rather than constantly in the background.

2.5 RCU Design

2.5.1 Hardware

The remote control unit is used to control the temperature within the room. The microcontroller used in the device needs to be capable of using a wireless connection to send and receive data. The microcontroller also needs to be battery powered and charged once per semester. After considering several options to achieve these goals, the best choice ended up being a power switch in the circuit. Since this requires human intervention, and we needed to be able to measure the temperature in the room more periodically, we decided to move the temperature sensor to the MCU. To show the relevant data on the RCU, we needed a display that could show the 4 digits for the temperature, and could be easily connected to the microcontroller. The unit also needs a method for the user to input the desired set temperature.

2.5.2 Software

The MCU provides data to the RCU via TCP sockets, where the RCU acts as the client in the scheme. In order to propagate the set temperature information to the WCU, the MCU writes the requested set temperature to the database. When an error is reported on the steam valve, the RCU will get an error code from MCU on start-up. It will display this error code instead of the temperature information and prevent the user from entering new temperature requests.

2.6 WCU Design

2.6.1 Hosting

The web control unit is designed to satisfy our clients requests for remote and mass control of the system. We determined that the most accessible medium to handle this would be a website. Using a website, users with an internet connection and the correct permissions are able to view and control temperatures. However, we didn't want absolutely anyone to have access to the site, so we decided to have it hosted on an ISU domain, and which restricts access to ISU network connections only.

2.6.2 Database

Creating a database lets us store vital data about the system in a central repository, accessible to the WCU and MCU. Each MCU reads and writes to the database; the WCU is able to access this data without having to directly connect to the MCU. Instead of referring to the MCU by its room number, we use a block reference number. This is to deal with the issue that multiple rooms are controlled by a steam valve in a different room. Users in a room without a steam valve might have trouble identifying which MCU to adjust, if it were identified only by room number. The database maintains a table with the room numbers corresponding to the block, which is also displayed on the website.

2.6.3 Functionality

Users need to be able to view and update the temperature for a single valve, control all valves at once, and have the ability to handle errors. The website essentially acts as a front-end interface to the database. This keeps the overall design as modular as possible. Users are able to find the correct MCU to update, change the set temperature, and view the current temperature. They are also able to perform mass control of all of the valves, and resolve errors. All updates to the site are pushed to the database, so that when the MCU goes to adjust the temperature, it can see any recent changes made by the WCU.

2.6.4 Errors

The website also displays the current errors reported from the valves and provides a page to mark the errors as resolved. The client needs to be notified of errors via a standard and a reliable service. Errors also need to be reported as soon as possible. All errors are logged in the database, so the error notification script has to have access to it. It also needs access to an emailing utility that can be automated. Because a possible error is network connectivity issues from the MCU, we need to host the error reporting program on the web server. This mitigates a scenario where errors aren't reported, because the MCU can't connect to the internet. Our program will need to run periodically on the web server to check for errors logged in the database and sends a message with the error information.

3. Implementation Details

3.1 MCU

3.1.1 Motor and mount

We choose to use the IG42 24VDC 078 RPM Gear Motor with Encoder as it provided 18 kgf-cm (~15in-lb) of torque which is significantly more than the valve required. This motor was repurposed from the previous project's supplies, so it did not count against our budget. Additionally, the built-in encoder allows the software to monitor the motor's rotation. The motor mount was designed using OnShape, a free, collaborative, online CAD software. The motor mount is machined from a single piece of angle aluminum using a CNC mill. To assist with alignment and account for variations between valves, we chose to put slotted holes in the angle aluminum so that the motor could be slid laterally during installation. The mount is attached to the valve using a commercially available pipe bracket which is rated for steam piping. The green plastic insert comes in various pipe diameters; which allows the mount to be used on different sized valves simply by changing the plastic insert. The motor is coupled to the valve stem with an aluminum coupler which uses set screws to create a temporary hold on the motor output shaft and the valve stem.

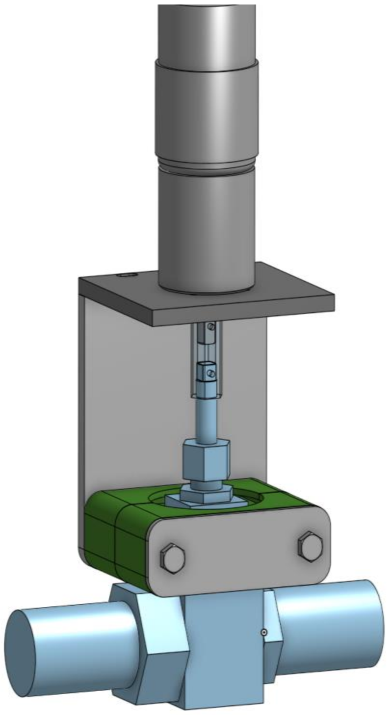


Figure 4. Motor mount

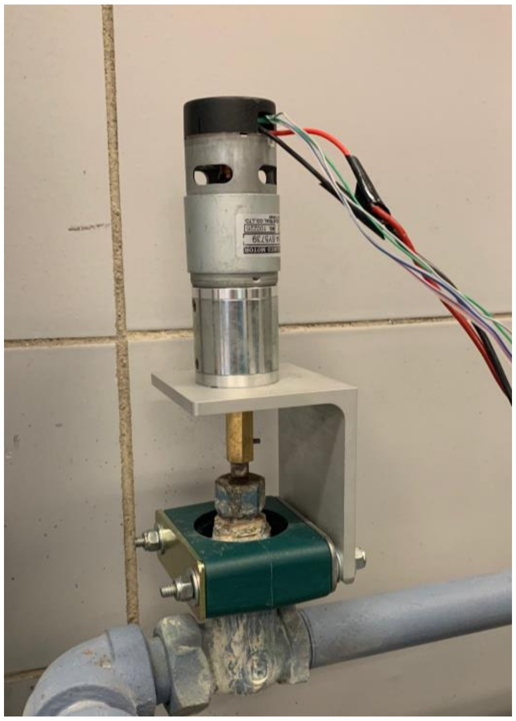


Figure 5. Final motor mount design

3.1.2 Hardware

The Raspberry Pi 3 B+ was the best option for the microcontroller on the MCU. It had the correct amount and type of pins for the functionality we needed. The chosen temperature sensor is the MCP9808 High Accuracy I2C Temperature Sensor Breakout Board from Adafruit; the Pi supports the I2C connection. The Raspberry Pi alone could not control the motor, so we used an Adafruit DRV8871 Breakout board which could interpret a PWM signal from the Raspberry Pi. The encoder from the motor is also connected to I/O pins on the Pi, so that they can be accessed in software. We created our own custom hat for the components that needed to connect to the Pi. This hat is equipped with both a MPC9808 temperature sensor and DRV8871 DC motor driver. During our initial testing, we used Adafruit's breakout boards for for both of these chips. For our final hat, we used the open source schematics Adafruit provided for these breakout boards and recreated both designs on our hat with additional logic for I/O pin connections to the Pi. This provided a much cleaner product then having both breakout boards sitting on top of another PCB on top of the Raspberry Pi.

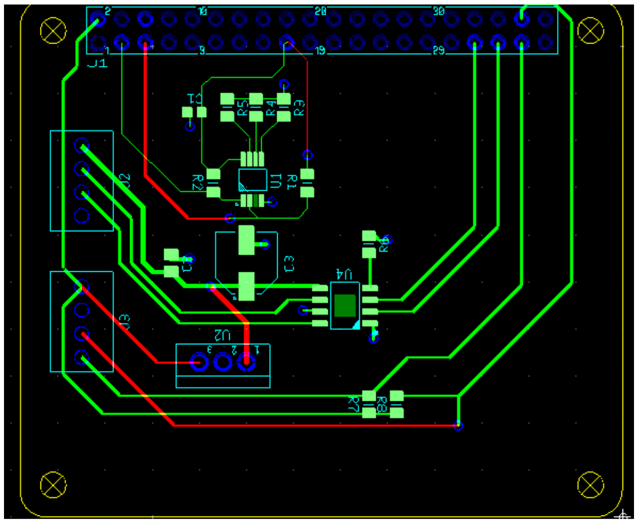


Figure 6. MCU PCB

The MCU has two unique power supplies connected to the custom hat: A 24V DC power supply and a 5V DC power supply. The 24V power supply is used to power the motor. The 5V power supply is used to power the Raspberry Pi. The Raspberry Pi has additional I/O pins that are used to power the MCP9808 and DRV8871.

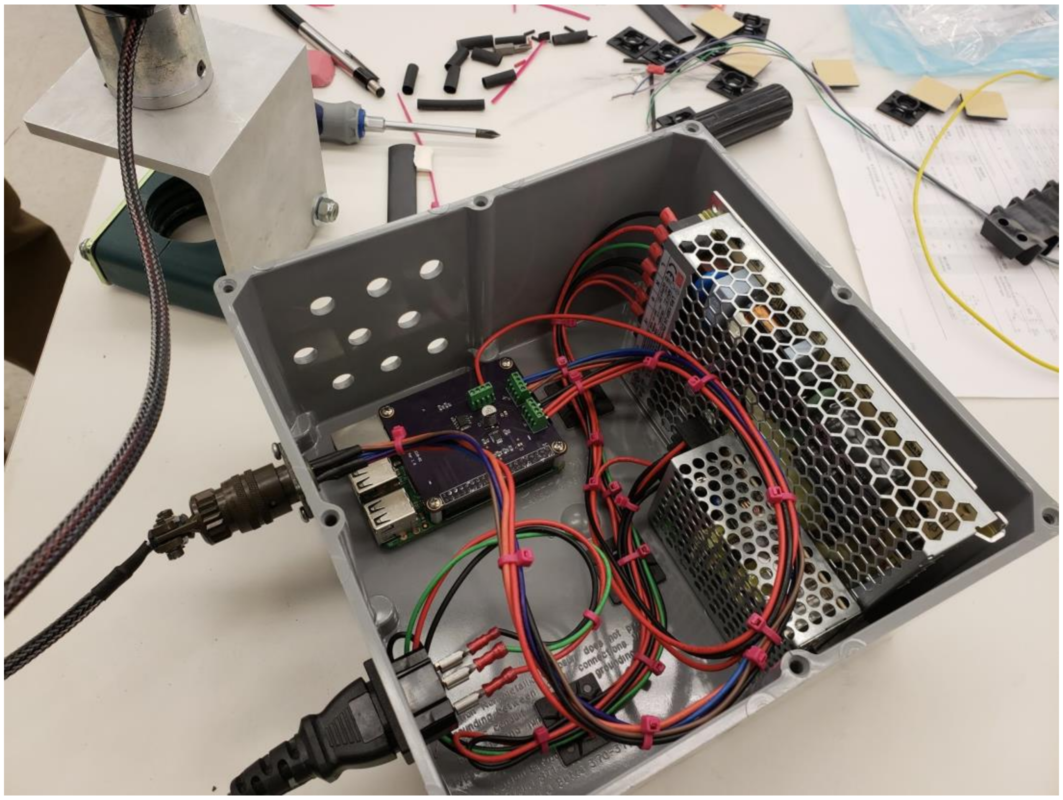


Figure 7. Assembled MCU and packaging

The Raspberry Pi also provides plenty of processing power to handle running the software, as it supports multithreading and has on-board wireless LAN. The Pi able to connect to Iowa State University's network, which enables the connection to the RCU via TCP sockets and the SQL database.

3.1.3 Software

All software on the MCU was written in Python. This language provides the most flexibility and ease of use for connecting to the components and network. The speed of execution is also sufficient for this project. The code that runs periodically is kicked off by Cron, which is configured to execute the adjust_valve.py and update_database.py at 15 minute intervals. Cron is a package used by the operating system to schedule custom jobs. The other script, wifi_comm.py, is added to pm2, which handles the operation as a background process by the operating system and runs the script at startup.

3.2 Remote Control Unit (RCU)

3.2.1 Hardware

The microcontroller chosen for the RCU is the Adafruit Feather Huzzah, which includes an onboard ESP8266 WiFi chip. The battery we chose to use is an Anker PowerCore 5000mAh External Battery, because it has sufficient charge capacity, charge indicator lights, is easy to connect to the microcontroller via the microUSB port, and can be recharged through a USB port. The Feather supports the I2C standard, enabling us to use a variety of screen types with ease. Since the battery consumption was no longer as major of a concern, we decided to use a quad-alphanumeric LED character display that had built in I2C. This allowed us to use fewer pins than an LCD display, which was necessary because the microcontroller only has 9 GPIO pins. The temperature can be controlled using buttons that are momentary switches. They are easy to interact with, use little power, and are inexpensive. The WiFi chip also lets connect to the MCU via TCP sockets.

The PCB for the RCU was designed to be a hat for the Adafruit Feather Huzzah using the design files Adafruit published online for measurements to make sure the hat would fit. The hat utilized male I/O pins on the bottom of the PCB to fit the female headers on the Adafruit Feather Huzzah. It also routed the appropriate traces to a five pin header to attach the LED display. Lastly, it had a place for the two buttons that would change the desired temperature up or down.

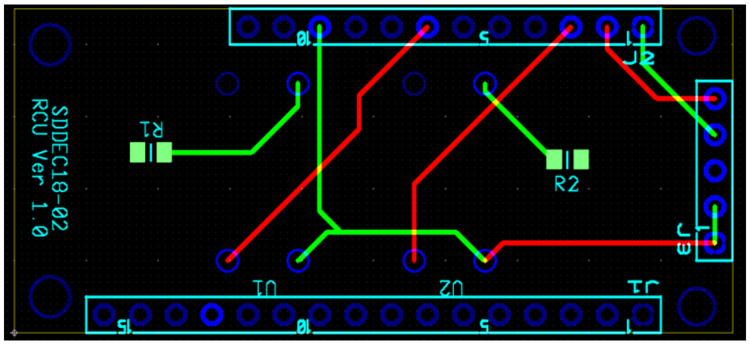


Figure 9. RCU PCB

3.2.2 Software

The RCU is flashed with a single program. It starts off by connecting to the internet, and then connecting to the MCU via the TCP socket. Once the MCU accepts the connection from the RCU, it sends the display data to the RCU. This data will either be the current and set temperatures, or an error code. This data will be printed to the LED display. The buttons are configured as inputs and use switch debouncing to ensure the temperature is changed only once with each button press. The user has 30 seconds to press the buttons and make changes to the temperature before the socket connection expires. The software limits the options the user can select to between 60 and 80 degrees. Before it expires, the RCU sends an update to the MCU. If the user did not change the temperature, "time" is sent back to the MCU, to represent a timeout (no change). Otherwise, the new temperature is sent to the MCU. On timeout, the display prints "SENT," to indicate that the change has been submitted. The user is then supposed to turn the power switch to off. If the user does not flip the switch, this message displays for around 30 seconds before the RCU automatically resets the connection. If the RCU received an error code from the MCU, it will display "ErrN" where N is the error code. The buttons will also be disabled if an error was received, so that the user cannot enter a new temperature.

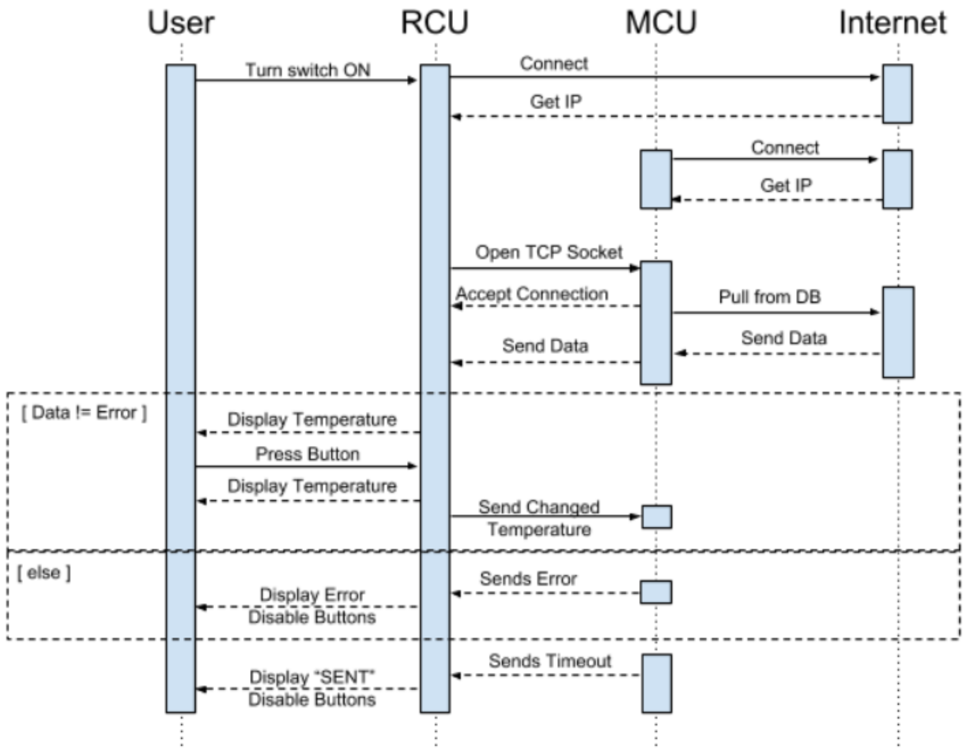


Figure 10. Communication sequence diagram

3.3 Web Control Unit (WCU)

3.3.1 Hosting

The remote control of the MCU has been implemented as a website. This site is hosted on the internal ece.iastate.edu network and domain, which is hosted and maintained by ETG. The website was designed using the Java Spring Framework, which uses an Apache Tomcat server, and Thymeleaf template rendering for HTML web pages. The database was also created and is hosted by ETG.

3.3.2 Database

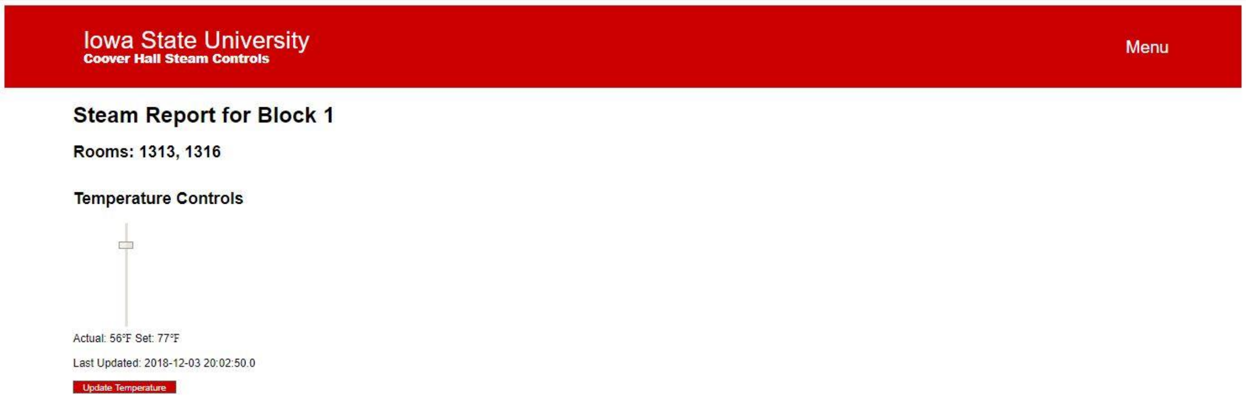
The database was created in MySQL, because it is robust, scalable, and easy to connect to in Java (from WCU) and Python (MCU). The tables were created through the MySQL Workbench. We use SSL certificates when making connections to the database, which encrypts the connection and data, giving our project an extra level of security.

The database consists of two tables, a Room table to track which rooms are connected by a single valve, and a Block table that contains the reports for every Block in the system. The Room table contains a list of Room_ID and Block_ID pairs. The Room_ID is the number that the room is referred to as in the Coover Hall floor plan. The Block_ID is a number from 1 to the number of Blocks in the system used to identify a group of rooms controlled by the same valve.
As stated above, the Block table contains all reports for each Block with the following format:

- Report_Num - An auto-incremented integer that is unique to each report
- Timestamp - Time/Date for each report "Year-Month-Day Hour:Minute:Second"
- Block_ID - Number of the Block represented by the report
- User_ID - String of the user that sent the report
- Current_Temp - Current temperature in the room that contains the valve
- Set_Temp - Desired temperature for the room that contains the valve
- Error_Code - Number from 0 to 9 reflecting the current error state of the valve

3.3.3 Functionality

The main page contains the floor plans for Coover, with a link on each room that directs the user to the relevant Block page. Within each Block page, the user can view the time of the last reported state of the valve, as well as the current and set temperature for the room containing the valve. They can also adjust the temperature by moving a slider to the desired temperature, which sends an new report to the database. The slider limits the temperature options to between 60 and 80 degrees. Additionally, if the block is currently experiencing an error, temperature controls will be disabled.



Steam Report for Block 1

Rooms: 1313, 1316

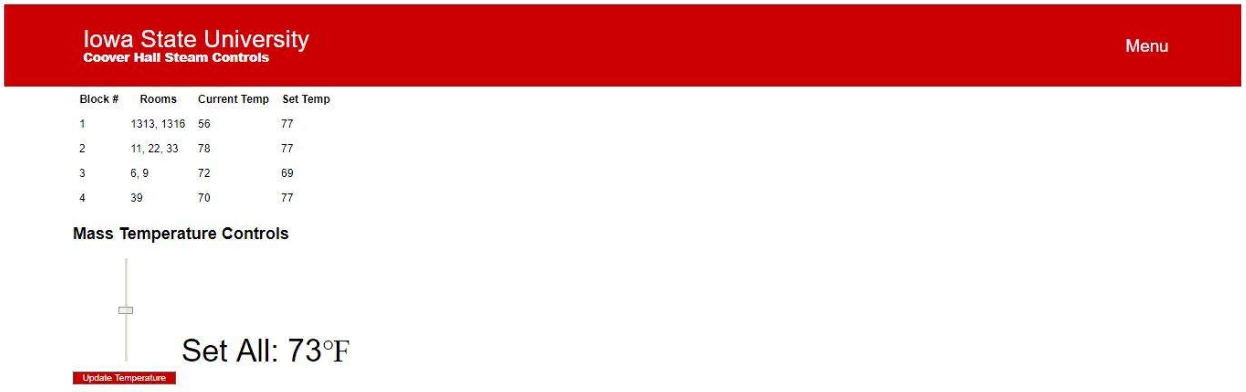
Temperature Controls

Actual: 56°F Set: 77°F

Last Updated: 2018-12-03 20:02:50.0

Figure 11. Block report

Mass control of all valves works in the same manner as an individual Block with the exception that an update is sent to the database for every Block on the list. If any block is currently experiencing an error, temperature control will be disabled for that specific block.



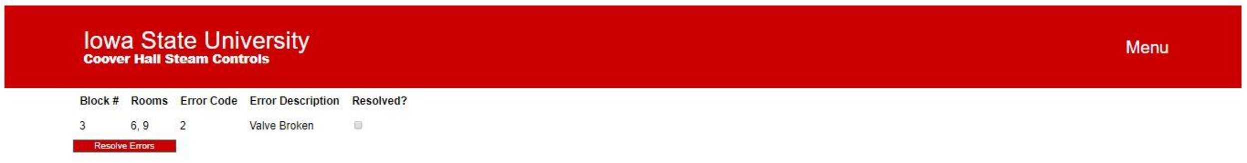
Block #	Rooms	Current Temp	Set Temp
1	1313, 1316	56	77
2	11, 22, 33	78	77
3	6, 9	72	69
4	39	70	77

Mass Temperature Controls

Set All: 73°F

Figure 12. Mass control

The error resolution page displays any Block with an existing error, along with the code and description for that particular error. The user is able to mark each error as resolved when the error is fixed, which will update the database with a revised report.

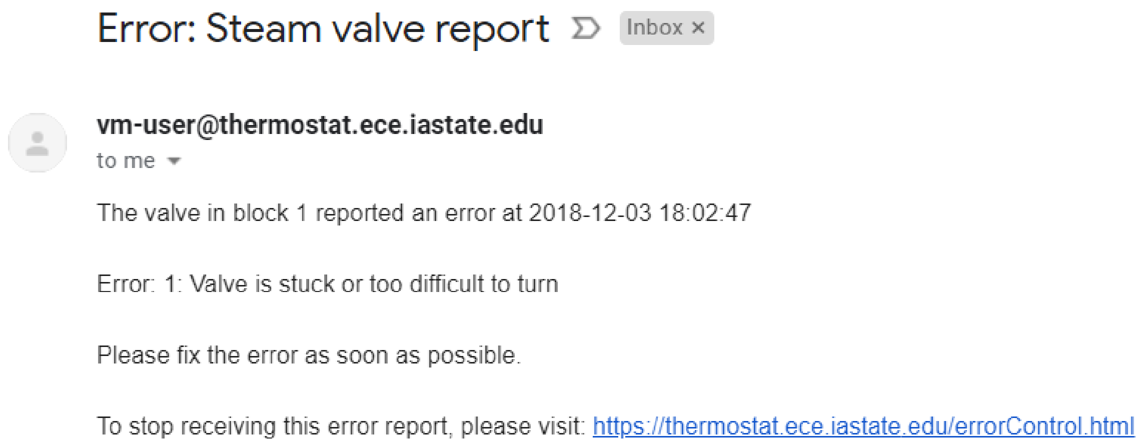


Block #	Rooms	Error Code	Error Description	Resolved?
3	6, 9	2	Valve Broken	☐

Figure 13. Error Resolution

3.3.4 Errors

The error notification script on the web server is configured to run periodically
In the event that the MCU fails to connect to the internet and its responsible for sending an error, then the client will never receive notification that the MCU is down. Moving the error reporting to the web server mitigates this issues. In order to find and report the errors, they will need to be logged from a source that the web server can access. Email has been chosen as the method of alerting our client about errors, because it is standard, timely, and easy to automate.



Error: Steam valve report

vm-user@thermostat.ece.iastate.edu
to me

The valve in block 1 reported an error at 2018-12-03 18:02:47

Error: 1: Valve is stuck or too difficult to turn

Please fix the error as soon as possible.

To stop receiving this error report, please visit: https://thermostat.ece.iastate.edu/errorControl.html

Figure 14. Error notification email example

4. Testing Process and Results

4.1 Functional Testing and Results

4.1.1 MCU

Since we decided to reuse the motor and encoder from the previous group's attempt, we needed to verify the motor still worked. We also needed to ensure the motor controller circuit we designed would work with the components. We hooked up the motor to one of the power supplies in the labs and gave it varying voltages and evaluated the response. We also monitored the output of the encoder with the oscilloscope to verify it was working as expected. Once the motor's operation was confirmed, we attached the Raspberry Pi and attempted to control the motor from it. The code on the Pi sent a PWM signal to the motor controller which determined the speed of the motor.

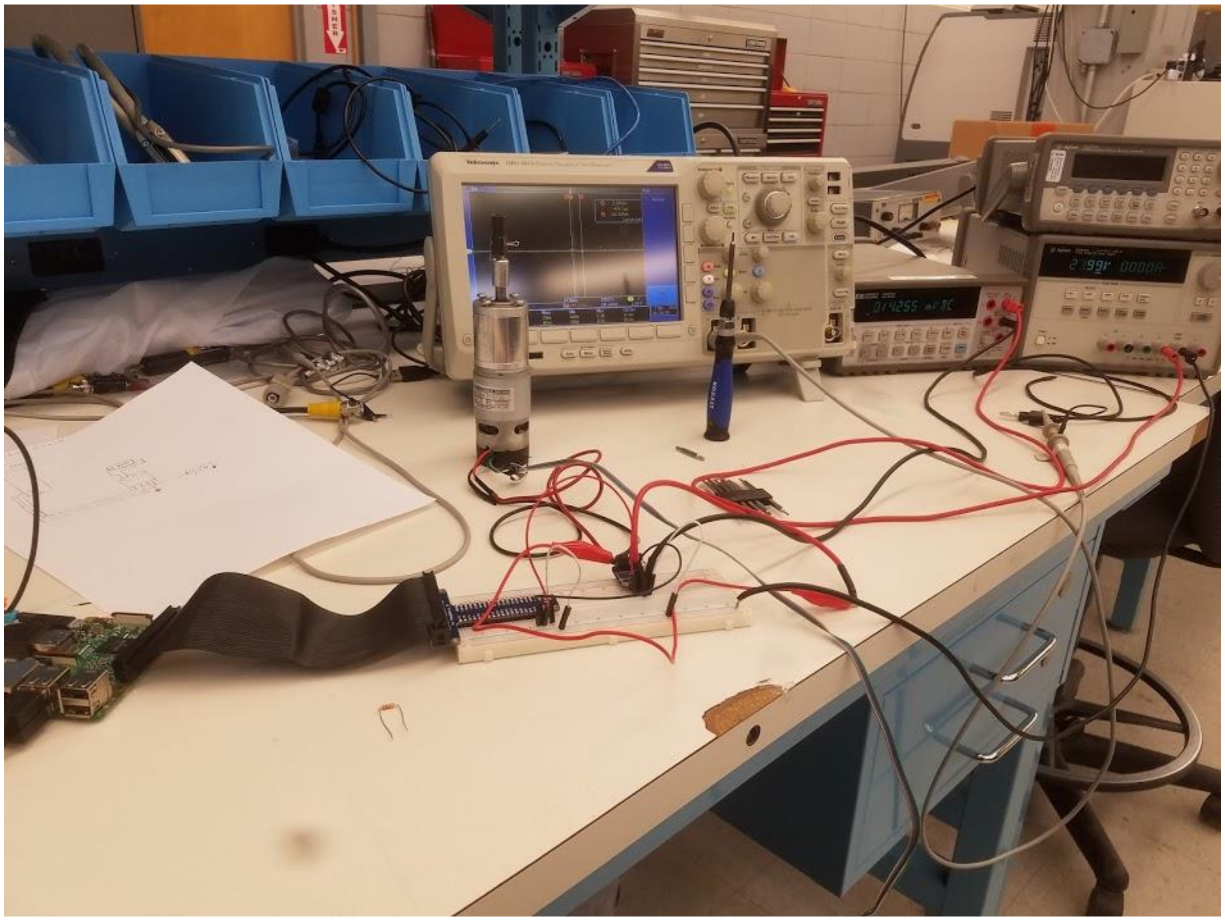


Figure 15. Testing the motor and encoder with the Raspberry Pi

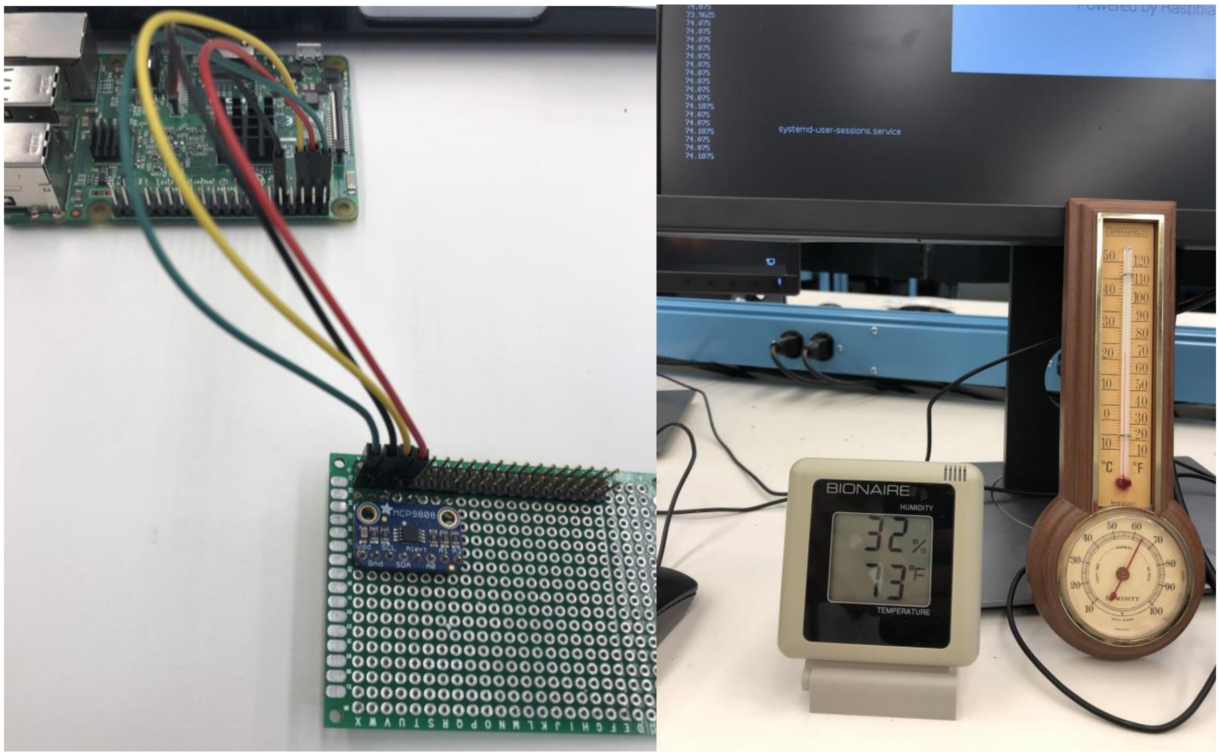
We also tested to make sure the Pi could read the data from the encoder. The encoder provides a square wave to one of the I/O pins of the Pi. A full rotation of the motor shaft corresponds to 420 high sides of the square wave. This test confirmed that we could monitor the angle of rotation of the motor in software.

4.1.2 RCU

The code on the RCU was tested to ensure it could connect to the MCU on-demand, errors were reported when present, and data was interpreted and handled correctly. We measured the connection to the MCU by starting the server code as a background process on the MCU, and then repeatedly making connections to the RCU. We let the connection between the two end: gracefully (wait for "SENT" and power off), ungracefully (don't wait for "SENT" and power off), and timeout/reset (wait for "SENT", software attempts a reconnect, and then power off). We then attempted a reconnect. In all cases, we saw the RCU able to reconnect to the MCU and transmit data successfully.

4.1.3 Temperature Sensor Testing

We decided to use the Adafruit MCP9808 Precision I2C temperature sensor. We wanted to verify that this sensor would work accurately with our Raspberry Pi. We connected the sensor to the Pi and used a Python script that printed the current temperature to standard output. We then compared that value to dedicated thermometers and ensured that the value was within 2 degrees of the actual temperature.



```
73.9625
74.075
74.075
74.075
74.075
74.075
74.075
74.075
74.075
74.075
74.075
74.1875
74.075
74.075
74.1875                    systemd-user-sessions.service
74.075
74.075
74.1875
```

Figure 16. Test setup to measure temperature sensor accuracy

This testing was repeated when the MCU hat was attached to the Raspberry Pi. In this configuration, the temperature sensor was placed approximately 2 cm from the Pi CPU. We recorded the temperature in the room as well as the temperature reported by the sensor over the course of 3 hours; the Pi performed its normal operations during this time period.

Accepted Temp (Fahrenheit)	Recorded Temp (Fahrenheit)	Difference (Fahrenheit)
71	87.8	16.8
69	85.6	16.6
71	91	20
73	94.5	21.5
74	96.3	22.3
69	86.5	17.5
70	89.3	19.3
72	90.3	18.3
75	93.3	18.3
73	89.825	16.825
72	88.36	16.36
71	88.1	17.1
75	93.2	18.2
72	87	15
68	83.075	15.075
70	85.5	15.5
	Results	
	Average	17.79125
	Max. Deviation	4.50875

Figure 17. Temperature sensor testing results from final placement

The results are unsatisfactory. The difference in the actual temperature and the recorded is almost proportional, but the amount of deviation seen in this small snippet is enough to be quite unreliable for our system. We would not be able to use this current temperature data in our control of the system.

4.2 Non-functional Testing and Results

4.2.1 Battery

In order to verify that the battery lasts all semester, we wanted to test how many power cycles the battery would support. We assumed the normal operation of thermostat would be approximately 1 change per day that class is in session for an entire semester. That comes to approximately 90 connections. Assuming the adjustment of the temperature takes roughly 30 seconds, we wanted the battery to last for at least 90 minutes. The battery that was chosen has three indicator lights that show the approximate life left (all 3 lights on correspond to between 50 and 100% battery life). The procedure for the test was as follows:

1. Turn on the MCU
2. Turn on the RCU
3. Change the set temperature
4. Wait for the change to be pushed on the database from the MCU
5. Turn the RCU off
6. Repeat 2-6 until 90 minutes or battery is at 50% (only 2 lights showing)

During this test, we found that the average time to change the temperature from turn on to updated server was 42 seconds. We had 149 power cycles in just over 105 minutes. All three indicator lights were still on at the end of this test. Therefore, we can conclude that the battery will last for one semester or more when used as designed.

4.2.2 Error notifications

Our client wanted errors reported as soon as possible. Since we elected to use email, we were concerned that there was a potential for the reports to be queued and sent later than intended. Specifically, the email needed to reach our client within 30 minutes of being reported. The web server runs the script to check for errors every 10 minutes. In our setup, we simulated errors by injecting them into the database. We evaluated the latency between the time the error was reported and the time the email showed up in our mailbox. The greatest latency we saw was 13 minutes. If the error reporting check happens 10 minutes after the error, and the latency is the greatest that it can be, at 13 minutes, then we are confident that the response time will be within the 30-minute window reliably.

5. Related Products

Our group went online to find other products on the market that are similar to our steam valve retrofit. The closest two products we could find were a Danfoss temperature sensor and a Belimo valve retrofit product. However, neither of these products offered a web interface for monitoring and controlling room temperature. There are several commercially available “Smart home” thermostats that can be controlled remotely, however none of them provide the capability to integrate with a custom valve retrofit.

6. Final status

6.1 MCU

We were not able to complete temperature control testing. This would have entailed setting the temperature in the room and evaluating the motors response and ability to hold the temperature within the margin specified by our client. The software, circuitry, and packaging is considered complete up to this point, but we are confident changes would likely need to be made to the motor controller software based on the testing we were not able to complete. The temperature sensor would also need to be moved outside of the packaging to give more reliable and consistent data.

6.2 RCU

We completed the circuitry and software to a workable, proof-of-concept level. However, we were not able to create packaging for this device, nor were we able to secure the connection to the MCU. The software would need to be updated to support encryption with the MCU.

6.3 WCU

The site design is complete and functional and the emailing capability is also complete. We were not able to provide authorization of users though, which would not be ideal for deployment. To get authorization via LDAP, the webcode would need to be rebuilt with Spring Security enabled and configured to use ISU's LDAP. With this turned on, the web server would need to be configured to use HTTPS. This encrypt the passwords used to connect via LDAP.

6.4 Integration Testing

Since the MCU was not tested to control temperature, final integration testing would need to be performed on the entire system before deployment. While we were able to independently verify the behavior of all other components, long-term testing of the entire system is necessary.

Test	**Measures**	**Procedure**
Valve retrofit physically fits on & can control selection of valves	Best case (some wall with no interference) and worst cases (corner, tight fit, etc)	Place the MCU on the selected valves and see if it fits and works
Positions valve to optimal location based on temperature	Temperature variation between 2 degrees of set point	48 hour test, record changes and set vs actual
Error reporting triggers correctly, accurately, and timely	Email sent within 30 min of error found; error doesn't persist	Simulate or create an error and measure the response of the system
Usability	RCU and WCU not 'difficult' to figure out	Ask advisor and a member of faculty to use them and report their reaction
RCU consumes battery life as expected in testing		Calculate expected battery consumption and compare to actual after tests
WCU correctly secured	Only accessible to admins and professors, HTTPS enabled, connection to database through SSL	Verify HTTPS on connection, check access with selection of admin, faculty, and students, and ensure database connection through log files
MCU chooses correct temperature from the database	Last input temperature is the one chosen	Add some temps, see that the MCU uses them as set

Figure 18. Proposed integration test cases

6. Appendix I - Operations Manual

6.1 Setup

6.1.1 MCU

1. Insert the sd card into a Linux machine (using adapter if necessary). Find the sd card by running

   ```
   sudo fdisk -l
   ```

   before and after inserting the disk. Once you have the path to the disk, for example /dev/sda1, run the command to image the disk (the path to the image on the Linux machine is shown below as /home/user/Documents/steam_valve.img), changing the image and device path as necessary:

   ```
   sudo dd bs=1M if=/home/user/Documents/steam_valve.img of=/dev/sda1
   ```

   After the command is finished imaging the sd card, insert the card into the Pi, hook up an HDMI cord, power cord, keyboard, and mouse. Login using the following credentials:

User: pi
Password: buttersidedown

2. Run the commands to setup the network:
   ```
   ifconfig wlan0
   ```
   Find the MAC Address after ether
   Ask ETG for a static IP for that MAC address. Once the IP is assigned and registered for the network, reboot the Pi.
3. Login again and run the command:
   ```
   startx
   ```
   Open a terminal using the black square icon in the upper left hand corner. Run the command for the startup script:
   ```
   ./startup.sh
   ```
   Following the prompts to setup the scripts.
   Shutdown the Pi using:
   ```
   sudo halt
   ```
4. Remove the valve handle. Mount the motor on the pipe, aligning the motor shaft with the valve shaft. Once aligned, tighten the bolts attaching the motor to the pipe to maintain alignment. After adjusting the motor, tighten the set screws on the motor shaft and the valve shaft. The wiring diagram is shown below.

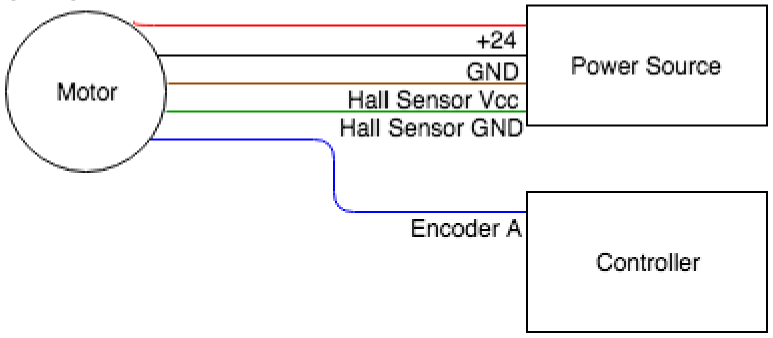


Figure 19. MCU packaging wiring diagram

After hooking up the wires as shown above, mount the control box to the wall and connect to power.

6.1.2 RCU

1. Change the ssid in line 10 of Wifi_Comm_Working.ino script to represent the name of the WiFi network you want to connect to.
   ```
   const char* ssid = "Name_of_Wifi_Network";
   ```
2. Change the host value to represent the static IP address of the MCU received from the ETG. This is done on line 15 of Wifi_Comm_Working.ino script.
   ```
   const char* host = "Static_IP_of_MCU";
   ```
3. Get the MAC address of the Feather by uncommenting line 32 in the Wifi_Comm_Working.ino script. Ask ETG to give a static IP address for that MAC address.
   ```
   Serial.println(WiFi.macAddress());
   ```

4. Upload the script to the Feather board using the Arduino IDE.

6.1.3 WCU

No setup required.

6.2 Operation

6.2.1 MCU

No user operation necessary for the MCU.

6.2.2 RCU

1. Flip the switch to turn on the thermostat. Wait for the display to show the current and set temperatures. If an error code appears on the screen contact ETG.
2. Press the buttons to change the temperature.
3. Wait for "SENT" to be displayed on the LED display.
4. Flip the switch to turn off the thermostat.

6.2.3 WCU

1. Connect to the internet through the ISU network.
2. Navigate to thermostat.ece.iastate.edu

   Adjusting individual valve

      1. Select the room from the floor plan image or from the floor option drop down.
      2. Adjust slider value to desired temperature.
      3. Press "Update Temperature."

   Adjusting all valves

      1. Select "Mass Control" from the "Menu" dropdown.
      2. Adjust slider value to desired temperature.
      3. Press "Update Temperatures."

   Resolving Errors

      1. Select "Error Control" from the "Menu" dropdown.
      2. If any errors exist, they will be listed along with a checkbox.
      3. To resolve an error for a specific block, check the corresponding box for that block.
      4. Press "Resolve Errors."

7. Appendix II - Alternative and Past Iterations of Design

7.1 Prior group

This project is a continuation from another effort several years ago. Since the last project was from several years ago and had to meet different requirements, we decided to select parts of the project to adopt into ours, but did not elect to continue where they had left off. From the documentation left behind, that group had attempted to create their own controller board for the motor, that involved making a custom PCB for the microprocessor, ethernet and Xigbee network connections, LCD display, thermometer circuit, and power regulation. That team was not able to finish their PCB design, nor was there valve mount feasible for the operation of the motor. We were took elements from their design, as well as several physical devices, and used them in our approach.

7.2 RCU

One of the components that has undergone the most changes from our initial ideas is the RCU. We initially planned to used a Raspberry Pi in the RCU but had way too many unnecessary features that consumed way to much power. The second iteration of the RCU used a microcontroller that communicated via Bluetooth Low Energy (BLE). The assumptions made about the protocol turned out to be incorrect, and we determined that it would not be a feasible option for the type of functionality we needed. Since the Pi was critical to the MCU, and the MCU is critical to the project, we decided to change out the microcontroller to one that could communicate via WiFi. This microcontroller is used in the final product.

The microcontroller we picked out is the Adafruit Feather HUZZAH because it supported a featured called deep sleep. While the board is in deep sleep, it shut down the majority of its features. This meant that it would consume a very small amount of power which is what we wanted because it was on the battery that needed to last for a full semester. Our research online and initial tests measured the current draw in this state between 4 uA to 10 uA. With a 5000mAh battery, this was going to work fine. However, upon further testing we found this was not going to work. The way the board wakes up out of deep sleep is sending a signal from one of its I/O pins to the reset pin. We tried to use the buttons that control the temperature to also complete the circuit between the reset pin and the I/O pin using an OR gate so both buttons would work. The board would then have to be sending the turn on signal continuously which means the board is not actual in deep sleep. The current draw in this state was somewhere between 20mA to 30mA. This is still less then the current draw with the board is completely on but would still drain the battery in a matter of days which was unacceptable.

We then moved to trying to use a MOSFET as a switch between the battery and the Feather Board. The buttons were still wired up to the OR gate along with a signal from one of the I/O pins on the Feather Board and the output was connected to the gate of the MOSFET. This was not working. We discovered that the voltage output of the OR gate was not high enough to turn on the MOSFET we were using. By this point, we needed a solution we know was going to work as we still had a lot of work to do on the project in other areas. We scheduled a meeting with our advisor to discuss this issue we were having and proposed a dedicated switch to power the Feather Board. He approved the idea and we continued on with the next stages of the project.

7.3 Motor Mount

Another part of our project that went through several iterations before settling on a design is our motor to steam valve mount. Several different ideas where tossed around and 3D modeled during the early stages of the project. At first, we thought up a tower that clamped to the pipe similar to the clamps used in wood working to hold together two boards while glue is drying. We also considered using a PVC pipe to mount the motor as it would be cheap to buy and easy to cut. That was also scrapped. The second to last design was a metal box that would clamp to the valve and have a hole in the top that the motor would mounted in.

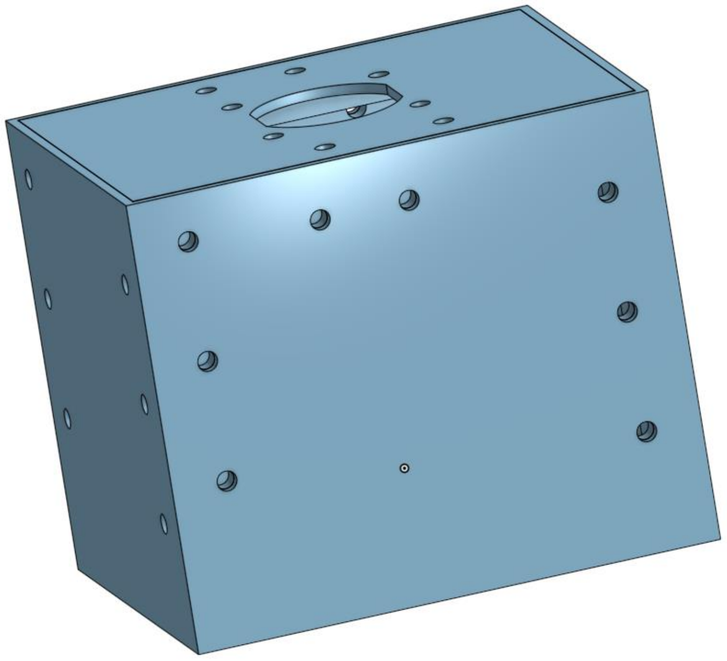


Figure 20. Box design for motor mount

We presented this to our client for manufacturing and determined this design is overkill for the motor we are using. He suggested that we only need one of the sides of the box to support our motor. That is how we come to our final design that was manufactured and shown below.

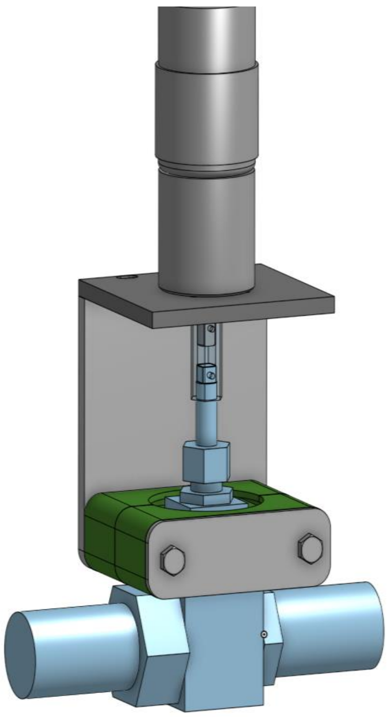


Figure 21. Final motor mount design

8. Appendix III - Lessons Learned

8.1 Security

We initially didn't plan on using any sort of extra security for the various database access we needed, but it was pointed out to us that it would be easily interceptable without using an SSL connection. ETG was able to generate certificates for this connection that we could use to connect from Python (MCU) and Java (WCU).

Access to the website was planned to be limited to only the staff in Coover Hall, ETG, and FPM. To do this, we started by trying to utilize Shibboleth, one of ISUs existing authentication systems. We later found out that we would be unable to use it for more than one semester, which is not ideal as a long-term solution. We changed our implementation to use LDAP (Lightweight Directory Access Protocol) as there existed some documentation within the Spring Framework on how to do so. This switch in authentication systems came along later than we had anticipated in the implementation process, and we were unable to dedicate time to get it to work.

We also determined that it would be advantageous to encrypt the connection between the RCU and MCU. We decided that the best approach would be to use AES-128 on each side to encrypt the data. After struggling to implement several open-source libraries, we also came to the realization that the encryption would be useless if we didn't also generate session keys. We looked into other open source libraries that provided the Diffie-Hellman Key Exchange protocol, but again, struggled to implement a solution that could be reliable. The common theme seemed to be that the libraries that supported the Arduino side implemented parts of the protocol in unexpected ways that proved to be difficult to interpret on the Python side from the chosen libraries. Again, this security measure was chosen late in the implementation phase, and the time spent trying to implement these solutions ended up blocking further necessary development.

We learned that it is important to consider the security of the software early in the design process, and to make finding the appropriate libraries and technologies to support it more of a priority. Further, even the smallest embedded parts of a project require security measures. If the item is connected to the network, we need to consider how that connection or the data can become compromised.

8.2 Temperature Sensor Placement

In our PCB design, we did not expect the processor to generate enough heat to affect the temperature sensor, but due to the placement of the temperature sensor in the middle of the Pi hat, as the processor warms up, the temperature sensor reports distorted data. We also realize that inside the control box, the airflow will be limited and therefore the temperature reading will be impacted to a greater degree. In future designs, we recommend placing the temperature sensor on the outside of the box.