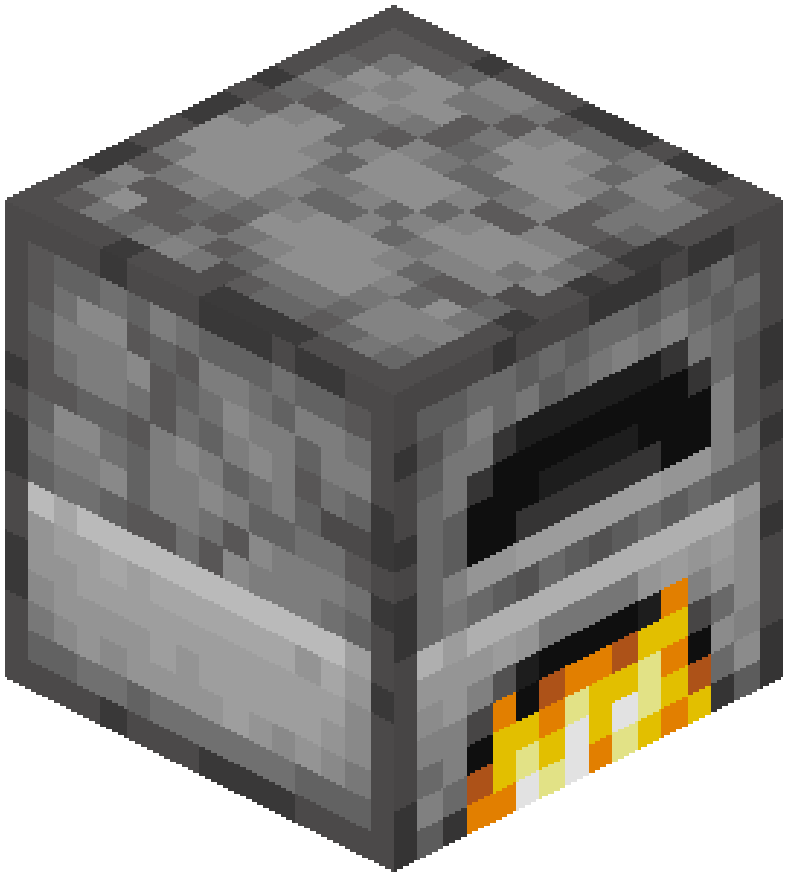
COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSM Hildreth: Furnace Controller

Team Members:

Colton Meyers
Josh Mendelsohn
Nathaniel Bujarski
Preston Yates

Revised June 17th, 2022



CSCI 370 Summer 2022

Client: Dr. Owen Hildreth

Advisor: Ms. Donna Bodeau

Table 1: Revision History

Revision	Date	Comments
New	May 16, 2022	Requirements section of final report created.
Rev – 2	May 17, 2022	Completed requirements section and team profile.
Rev – 3	May 18, 2022	Updated functional requirements based on newly discovered means of communicating with sensors. Added more key terms to Appendix A.
Rev – 4	May 19, 2022	Finalized requirements section before submitting current revision to advisor.
Rev - 5	May 27, 2022	Added system architecture diagram, wireframe, state diagram, and circuit diagram.
Rev - 6	June 3, 2022	Added software quality and ethics considerations. Update functional requirements and non-functional requirements. Added brief description before each architecture diagram.
Rev - 7	June 6, 2022	Updated functional requirements and definition of done.
Rev - 8	June 9, 2022	Began working on the results section.
Rev - 9	June 10, 2022	Completed software testing and updated test table. Added additional terms to Appendix A.
Rev - 10	June 13, 2022	Updated Introduction and Functional requirements. Started on Project Completion Status, Future Work, Lessons Learned, and Technical Design sections.
Rev - 11	June 14, 2022	Completed Completion Status, Future Work, Lessons Learned, and Technical Design sections.
Rev - 12	June 17, 2022	Final revisions

Table of Contents

I. Introduction

Our client, Dr. Owen Hildreth, is an Assistant Professor in the Mechanical Engineering Department at Colorado School of Mines. Dr. Hildreth's research focuses on nanometer to centimeter-scale additive manufacturing technologies and involves many mechanical and materials engineering principles such as mass transport, heat transfer, chemical kinetics, electrohydrodynamics, and corrosion [1]. For the variety of concepts that are applicable to Dr. Hildreth's research, there is an equally large variety of equipment in his lab that he uses to support his research. To simplify the handling of this equipment, Dr. Hildreth has designed and commissioned many custom MacOS applications to assist in the data collection and control for his lab equipment. However, there are still multiple pieces of equipment in his lab that use antiquated methods of control and data collection.

The goal of this project is to create a MacOS/iOS application to handle the operation and data collection for the furnace in Dr. Hildreth's lab. More specifically, there exist two mass flow meters, one for Argon gas and one for Nitrogen gas, and an overall furnace control box that we need to get values from and send commands to via our MacOS application. Up until now, Dr. Hildreth and his lab assistants have been manually updating each of the flow meters when they want to change gas flow rates, and they have been using a video camera to record the temperature values displayed on the furnace control box throughout their experiments. Since this method is very time consuming, Dr. Hildreth wants a Mac application that he and his lab assistants can use and maintain which will automate the process of data collection while also providing a simple user interface through which the flow meters can be updated. By utilizing an Arduino, it should be possible to communicate between the various sensors and the Mac application. The application should then be able to handle the processing, storage, and visual presentation of that data while also having logic for sending commands to the sensors. Overall, this application should help in streamlining communication between the furnace and the person who is using it.

II. Functional Requirement

- Arduino Setup
    - Communicate with two Gas Flow Meters simultaneously using 8-Pin Mini DIN cables and serial communication protocol RS232
        - Poll sensors for flowrate data
        - Send commands to update each sensor's flowrate
    - Read temperature values, expressed as analog voltages, from furnace's thermocouple
    - Conduct a status check to make sure that all sensors are connected
    - Send data received from the sensors to the Swift application using a USB cable
        - Stretch Goal: Send data via Bluetooth instead of USB
- MacOS Application
    - Communicate with the Arduino unit using one of the computer's serial ports
        - Detect, record, and parse data packets sent by the Arduino
        - Send commands of a specific format to the Arduino
            - Poll Argon flow meter
            - Poll Nitrogen flow meter
            - Set flowrate setpoint of Argon flow meter
            - Set flowrate setpoint of Nitrogen flow meter
            - Read temperature value
            - Check status of connection with sensors
    - Display data
        - Display current connection status
        - Display current values of temperature and flowrate
        - Graph data from sensors over time according to a preset rate
    - Save sensor data

        - Save temperature and flowrate data as a CSV file
    - Provide user interface for updating flowrates
    - Provide user interface for starting and stopping data recording

III. Non-Functional Requirements

- The MacOS application must be written using the XCode development environment
    - SwiftUI must be used to create the user interface as a MacOS application
- The Arduino program must be written using the Arduino development environment
- The DataGraph API must be used to implement the graphs on the GUI
- The GUI should not have any intrusive elements, such as alerts

IV. Risks

Table 2: Potential Risks

Risk	**Type**	**Impact**	**Resolution**	**Probability of Occurrence (1-5) 5 = very likely**
Sensor Malfunction	Hardware	The Arduino will not be able to get accurate values.	Re-write the Arduino with a new sensor	2
Wiring Short	Hardware	The Arduino will not be able to get any values from the sensor	Find the short and replace the wire with a new one	3
Arduino Failure	Hardware	Whole system will cease to function	Flash the Arduino with the code again. Re-write the new Arduino to sensors	1
Learning Swift	Software	Our code will not run correctly and get our desired outcome	Run tests and debug to make sure our code is semantically correct	5
MacOS Communication Error	Software	MacOS communication with Arduino devices fails	Gracefully handle the error in the software and indicate to the user to re-establish communication	3

V. Definition of Done

Our definition of done is that the MacOS application successfully records and graphs data coming from the Arduino, and that the application can communicate with the flow meters to set their flowrates. This requires the following items to be completed:

- The Arduino can communicate with and send data to the MacOS application
- The Arduino can read temperatures using a thermocouple
- The Arduino can read signals from the flow meter and send commands to it
- The MacOS application can send string commands to the Arduino
- The MacOS application must display graphs of measurements versus time
- The MacOS application must have a timer displaying experiment runtime
- The MacOS application must have fields which allow the user to update flowrate
- The MacOS application must be able to save all recorded data to a CSV file

VI. System Architecture

With regards to the hardware, both Apex Flow Meters, which measure and control flowrate, communicate using RS-232 serial. Since the Arduino uses a different voltage range than the flow meters, it is necessary to convert the RS-232 signals from the flow meters into TTL signals by using a MAX3232 converter. The thermocouple outputs analog voltages on the order of milli-volts, so it is necessary to amplify those signals using a MAX38155 thermocouple amplifier for K-type thermocouples. This amplifier outputs the information using SPI (Serial Peripheral Interface) to the Arduino. A USB connection is used to communicate between the Arduino and the MacOS application.

On the MacOS side of the project, the ArduinoController class acts as the interface between the Arduino and the application. This class communicates with the AppController class which handles most of the backend logic that exists within the application. AppController contains the timer object for creating the stopwatch object in the GUI and the timer object for determining when to send a data poll request to the Arduino. In addition to communicating with ArduinoController, AppController also communicates with the DataController and GraphController classes to interface with the file memory and DataGraph application, respectively. The DataGraph application is used to generate and format graphs which are displayed on the GUI. Finally, the GUI communicates with the AppController to convert user inputs into sensor commands and to retrieve and display data that is received from the sensors.

The overall system architecture on both the hardware and software sides is visually represented in Figure 1. The arrows indicate how various parts of the system communicate with each other. Edges without labels indicate that the connection is internal to the program, whereas edges with labels indicate the protocol which is utilized for communication.

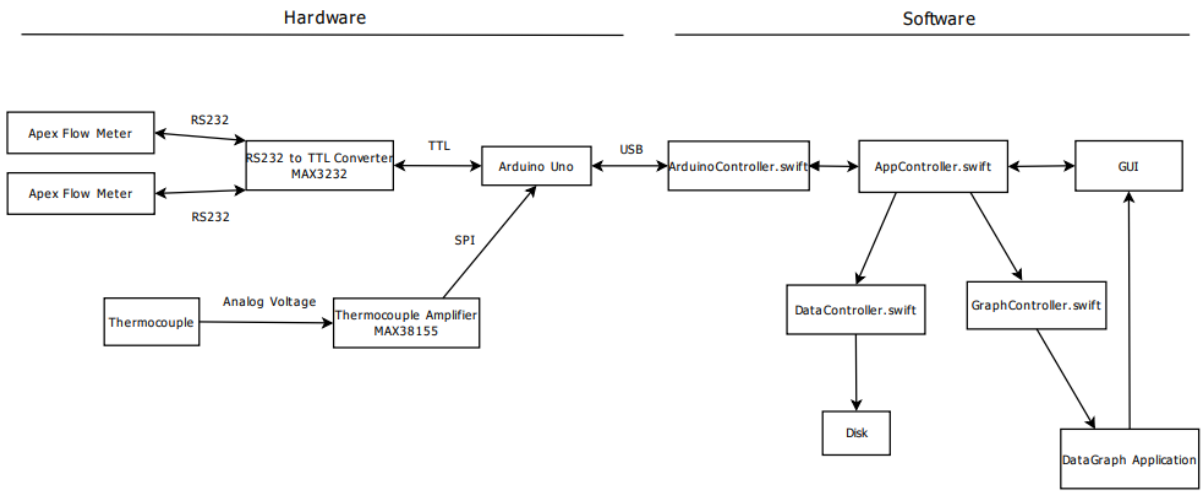


Figure 1: Furnace Controller Architecture Diagram

Figure 2 is the finalized graphical user interface developed in SwiftUI. It includes interfaces to start and stop the experiment and to control the temperature and flow rates of the mass flow sensors. Additionally, there is functionality to graph the gas flow rates and temperatures as a function of time. In the lower left corner, the "Minutes/Sample" text field allows the user to control polling rate of the sensors.

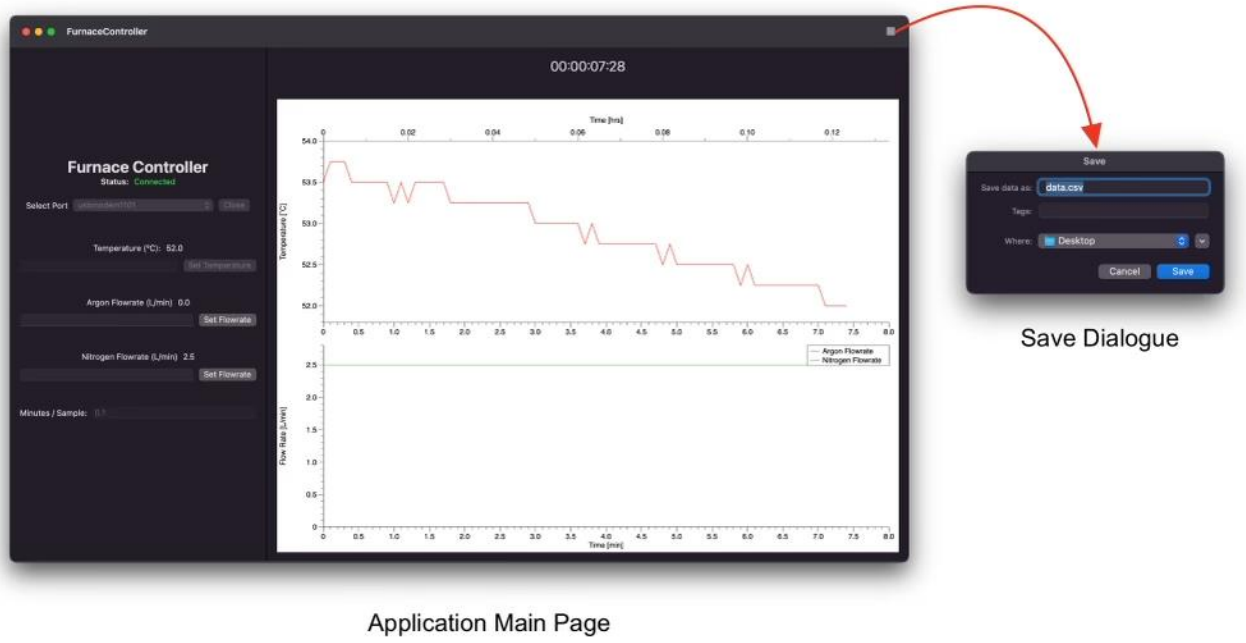


Figure 2: Furnace Controller Wireframe

VII. Technical Design: Sensor Communications

Introduction

The hardware of this project is crucial to our ability to communicate with the different kinds of sensors and get usable data to the MacOS application. The idea is that an Arduino UNO acts as a communications bridge between the

MacOS and the furnace sensors. The application talks to the Arduino over USB using a command response language, and the Arduino interprets those commands and sends back the requested sensor data. This simplifies connectivity on the Mac side of the project as only a singular USB port is needed while allowing the Arduino to do the job of controlling multiple sensors with its powerful GPIO.

Hardware Implementation

At its core, the Arduino interfaces with three sensor devices: thermocouple for reading temperature values inside the furnace and two Apex mass flow meters for collecting data about and controlling gas flow into the furnace. The Apex mass flow meter is a smart controller which can communicate using serial, collect gas flow data, and control gas flow. It has its own built-in language for commands and data responses and communicates using RS-232, a type of serial communication protocol. This serial communication protocol allows for the exchange of bytes which can be formed into ASCII text and interpreted by the Arduino program. RS-232 communication with the Apex flow sensors is achieved via a MAX3232 board which converts between TTL [0-5 volts] level voltages and RS-232 [-15 to +15] level voltages. Using this converter board, it is possible to communicate with the Apex flow devices and send arbitrary string commands for both fetching flow data and setting desired flow rates.

The other sensor is a type K thermocouple, which is a very simple sensor that uses the Seebeck effect to generate specific voltages at specific temperatures. These voltages can be read and converted into a temperature value. One limitation of the Arduino is that its analog to digital converter is too inaccurate to handle reading the voltage values generated by the thermocouple natively. To handle this issue, we employed a MAX31855, a thermocouple amplifier. This board amplifies the voltage generated by the thermocouple, converts that voltage to a temperature value in Celsius, and communicates that temperature back to the Arduino using Serial Peripheral Interface (SPI). SPI is yet another flavor of serial communication like TTL and RS-232 serial, however this protocol is synchronous meaning both devices share a common clock signal which dictates when bits are transferred. The result is still the same, a serial protocol which allows for the exchange of bytes which are formed into ASCII text and interpreted by the Arduino program.

Figure 3 is the circuit diagram for the Arduino and connected sensor components. It shows the aforementioned connection with the two Apex Mass Flow controllers through a TTL to RS-232 converter. It also specifically shows the pins utilized for the T1 (transmit line 1) and R1 (receive line 1) which are the specific hardware pins utilized to achieve communication with the Apex Flow Sensor devices. Since there are two pins utilized, communication is full duplex allowing for two-way communication at the same time. Another important observation is the Apex Flow Sensors are wired in parallel with one another. This is because both devices receive the command and then decide for themselves if they are the intended recipient of the command. This ambiguity is resolved by having each flow sensor device have a unit id which can be any alphabet letter A-Z. Figure 3 also shows the thermocouple amplifier and the external connection to the furnace thermocouple. It shows the three pins (CLK, CS, DO) which are essential to achieving SPI communication between the Arduino and the amplifier. Only one data pin is listed (DO) which indicates that this is a read only relationship between the Arduino and the amplifier. Finally, for clarity the USB connection with the Mac computer is not pictured, but pins D1 and D0 are reserved for hardware supported TTL serial communication with the connected Mac.

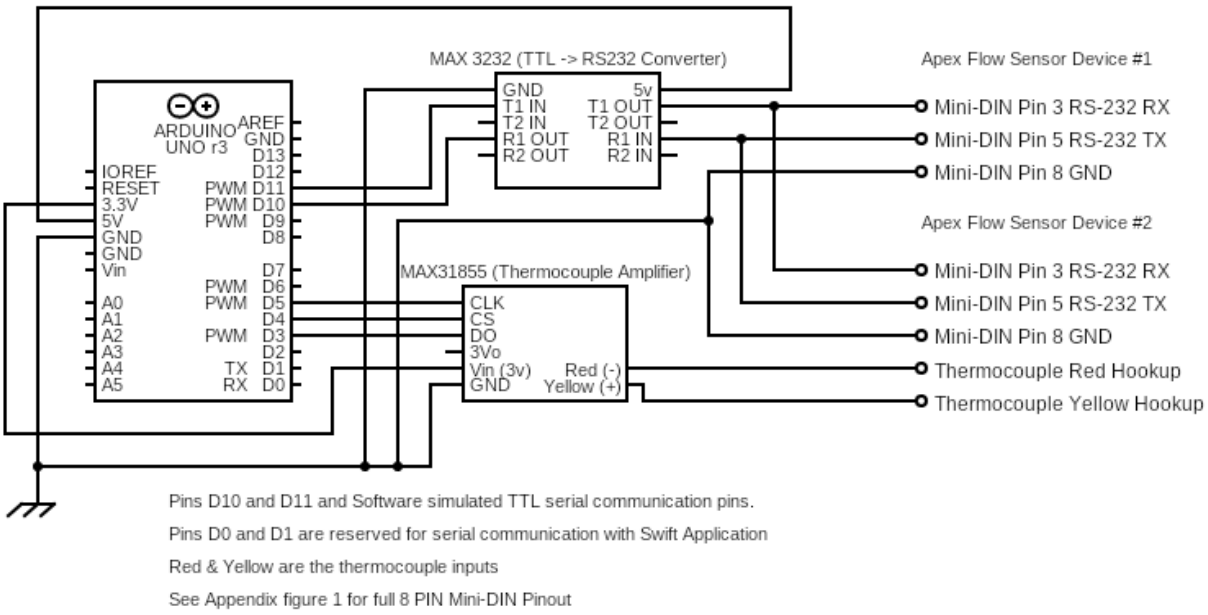


Figure 3: Arduino Control Board Circuit

Arduino Language

The Arduino has a command-based API which was created to allow for both manual testing and communication with the Swift Application. Communication is first achieved by connecting the Arduino to a USB port on a computer. Manual testing is conducted by utilizing a Serial monitor set to a BAUD rate of 9600. The following tables document the possibilities for commands sent as well as the expected responses in each circumstance. The API has proved extremely valuable as it allows for extensible control to be programmed into the Arduino with no concerns about regressing functionality due to the presence of a maintained format.

Command Format:

Each command has the 4 main components listed below. UID and DATA vary as described below. The command is always space delimited unless otherwise specified.

$ <UID> <DATA> ;

- **UID** is a positive 32-bit integer value
  - Every response to a command is given the same **UID** as the request had. This helps to map responses to requests and provide a mechanism for catching commands which have failed.
- **DATA** is a variable length string
  - It can either be an **Immediate command** or an **Apex Flow Sensor command**

Table 3: Immediate Commands

Command	Description
TEMP	Polls the thermocouple for the temperature in Celsius

STATUS	Prints status information about sensors.

Table 4: Apex Flow Sensor Commands

Command	Description
<Unit ID>	Polls the Apex Flow Device for a data frame about current flow values.
<Unit ID>s<floating point value>	Sets the setpoint (L/min target for gas flow) of the device to the value described.

- **$** is the beginning of transmission signal (BOT)
  - The BOT signal causes the Arduino code to transition to the read command state. In this state the Arduino fills an internal input data buffer with the received characters over the serial communication line with the MacOS application or serial monitor application (for testing).
- **;** is the end of transmission signal (EOT)
  - The EOT signal causes the Arduino code to transition to the execute command state. This concluded the filling of the buffer and allows for the parsing of the command now stored in the buffer. The resultant command then determines the next state transition. Further details on the state transitions can be viewed in figure 4.

Response Format:

$ <UID> <DATA> ;

- **UID** is a positive 32-bit integer value
  - This will take on the same value specified in the initial command
- **Data** a variable length string now containing the Data response from the initial command sent

Table 5: Response Data

Command Sent	Response Type	<DATA>
a	Flow sensor data frame	[Unit ID] [Absolute Pressure] [Temperature] [Volumetric Flow] [Mass Flow] [Setpoint] [Gas]
as<floating point value>	Flow sensor data frame with updated information about the setpoint	[Unit ID] [Absolute Pressure] [Temperature] [Volumetric Flow] [Mass Flow] [Setpoint] [Gas] If the change was successful, the setpoint will now reflect the floating-point value provided.
TEMP	Temperature value	<Temp in ºC>
STATUS	Debug info on current sensor connectivity status	OK – all sensors are connected and responding BAD <message> - one or more sensors are not responding

Error Response Format:

$ <UID> ERROR <explanation> ;

In many cases, communication cannot be established with a hardware sensor, or a malformed command is sent. The Arduino will respond with an error message describing the current issue encountered. Every response will conform to the pattern described above with explanation being a variable length string explaining the issue that was encountered. If a malformed command is sent without a UID then –1 will be assigned to the UID in the response helping to indicate this issue.

Arduino State Machine

The communication between the Arduino, the Apex Fow devices, and the MacOS application is all asynchronous. This means that the Arduino will receive bytes of data intermittently and over many clock cycles. This requires the ability to stay in a specific state for an undetermined amount of time while data buffers are filled. Previously in the Arduino language section the BOT and EOT characters were discussed which are the key components the Arduino waits for to signal the transition of states. Below is Figure 4, a finite automaton which visually describes these transitions of the Arduino code. The state machine proved to be an elegant way to handle the Arduino code as it allowed for the complexity of Asynchronous communication to be delegated to different states and helped add direction to the development of the microcontroller code.

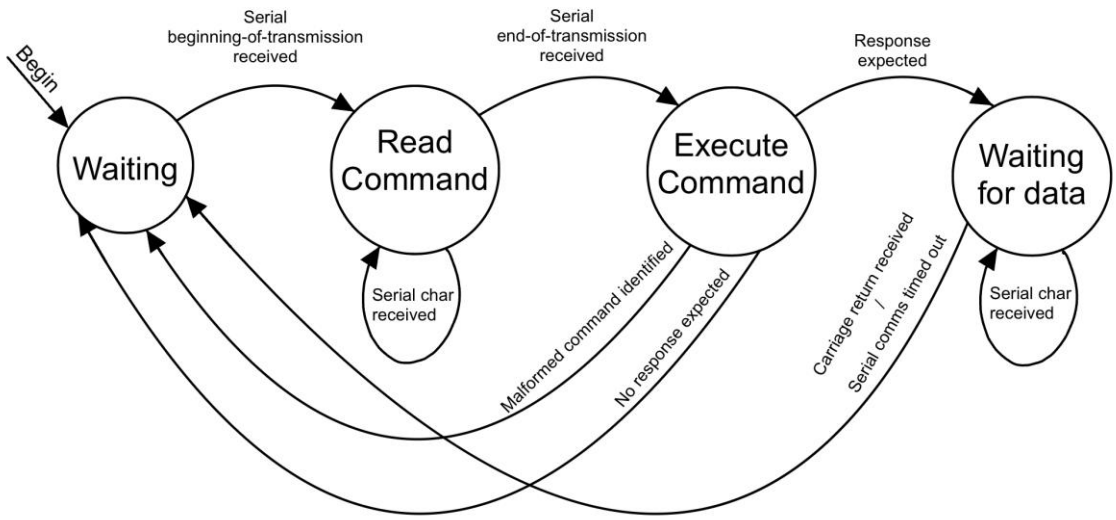


Figure 4: Arduino State Diagram

Technical Design Conclusion

Taken together, the hardware control boards, the Arduino state machine, and the Arduino command-based API allows for the simple control of sensors from the MacOS application and a useful implementation of the command response control format. Developing the command-based API proved extremely useful as it allowed for the decoupling of the Swift application from the Arduino microcontroller which simplified coordination, development, and testing.

VIII. Software Test and Quality

Software testing is essential to ensuring quality and usability of the designed application. The following tables detail the wide spectrum of tests that we covered in order to ensure usability and functionality for our program. It is important to note that due to time constraints, most of these tests are manual tests. User Interface tests are focused on UI functionality throughout the application and making sure intended functionality occurs when the application is in a specific state. Following that is functional testing which focuses entirely on the command-response API that we created

for the Arduino and verifying that the Arduino acts as expected when interfacing with the external sensors of the project. Finally, we have load testing which focuses on stressing the application and ensuring that the Arduino can handle high pooling rates, and the UI responds well to a user who is spamming buttons or otherwise stressing its interfaces.

User Interface Testing

Table 6: UI Tests

Test Name/Description	Action	Expected Result	Test Passed?
Buttons Disable When Bad Port Connection	Visually inspect the MacOS application when no Arduino is plugged in	Flowrate buttons and textfields are disabled (grayed out). Timer start button is also disabled.	YES
Connection Status Display Updates When Port is Connected and Disconnected	Select port with connected Arduino from dropdown menu and push open button in the MacOS application. Then visually inspect the connection status text. Push close button and visually inspect connection status text	When a port has been selected and the open button is pressed, the connection status should change from "Not Connected" to "Connected". When the close button is pressed, the connection status should change from "Connected" to "Not Connected". When the USB is unplugged, the connection status should change from "Connected" to "Not Connected". Edge Case: Status should remain as "Not Connected" and error message is given if Arduino is not connected to sensors.	YES
Port Selection Dropdown and Close Button Disable When Recording Data	Press start recording button in the MacOS application and visually inspect port selection dropdown and "Close" button	Port selection dropdown and close button disabled (grayed out) while recording.	YES
Flow Rate Values Automatically Update When Values are Out of Range	Enter numeric values greater than 1 and less than 0 into the flowrate textfields of the MacOS application and visually inspect resulting value in textfield.	If a value greater than 1 is entered (by pressing enter on keyboard or button on UI), the flowrate displayed in the textfield becomes 1. If a value less than 0 is entered (by pressing enter on keyboard or button on UI), the flowrate displayed in the textfield becomes 0.	YES

Flow Rate Does Not Update When Non-Numeric Values Entered	Enter non-numeric values into the flowrate textfields of the MacOS application and visually inspect values in textfield.	On entering a value, the textfield should not be updated.	YES
Minutes Per Sample Textfield Automatically Updates When Values are Below Minimum	Enter value below 0.1 into the textfield of the MacOS application and visually inspect state.	Upon pressing the start recording button, the value in the Minutes per Sample textfield is updated to 0.1	YES
Graphs Autoscale and Add Data According to Sample Rate	Start timer in the MacOS application.	Curve should be created on graph, and graph should auto scale as time goes on.	YES
Save Dialog Appears When Timer is Stopped	Stop timer in MacOS application.	Save dialog should appear	YES

Functional Testing

Table 7: Functional Tests

Test Name	Description	Action	Expected Result	Test Passed?
Flow Sensor Data Polling #1	Test if the Arduino can successfully get flow data from the Apex Flow Sensor with ID A.	Connect to Arduino with USB cable. Utilize a serial monitor with a 9600 Baud rate Send the command: $ 1 a ;	A data packet in the form $ 1 A +011.93 +038.53 -0.0000 -0.0000 +0.0250 Ar ; The values will vary, but there should be 8 fields. Including a unit ID (A in this ex), and a gas (Ar in this ex).	YES
Flow Sensor Data Polling #2	Test if the Arduino can successfully get flow data from the Apex Flow Sensor with ID B.	Connect to Arduino with USB cable. Utilize a serial monitor with a 9600 Baud rate. Send the command $ 1 b ;	A data packet in the form $ 1 B +011.93 +038.53 -0.0000 -0.0000 +0.0250 N ; Note this is device #2 The unit ID and gas vary accordingly.	YES
Thermocouple Data Polling #1	Test if the Arduino can successfully get	Connect to Arduino with USB cable. Utilize a	A data packet in the form	YES

	temperature data from the Thermocouple chip.	serial monitor with a 9600 Baud rate. Send the command $ 1 TEMP ;	$ 1 5 ; Where the second number, 5, indicates the degrees Celsius the thermocouple is reading.	
Flow Sensor Set Flow # 1	Set the flow rate for device A. This is to make sure the application can modify the flow rate.	Connect to Arduino with USB cable. Utilize a serial monitor with a 9600 Baud rate. Send the command $ 1 as0.025 ;	A data packet in the form $ 1 A +011.91 +037.50 -0.0000 -0.0000 **+0.0250** N ; The bolded number above confirms that the setpoint was set to +0.0250	YES
Serial Communications Timeout	Verify the Arduino will timeout if there is no response from the flow device. Within 3 seconds.	Connect to Arduino with USB cable. Utilize a serial monitor with a 9600 Baud rate. Send the command $ 1 c ;	An error packet in the form $ 1 ERROR serial communication with flow sensor timeout ; This is because there is no flow sensor device #3	YES
Malformed Command Error Handling	Verify the Arduino can identify that a malformed command is sent to it.	Connect to Arduino with USB cable. Utilize a serial monitor with a 9600 Baud rate Send one of the following commands: $ c ; $ 1 ;	An error packet in the form $ -1 ERROR malformed input ; This is because there is not a UID and Command element in the command.	YES
Sensor Status	Arduino	Send the command $ 1 STATUS ; To test the sensor status command, disconnect 0 or more sensors from the Arduino and re-run the command to verify a change In the status. For example, disconnecting sensor A will throw the sensor A relevant polling error.	A data packet in the form $ 1 BAD <INFO> ; <INFO> is a string identifying the issue It can be one of the following strings: Apex Flow sensor A did not respond when polled Apex Flow sensor B did not respond when polled Thermocouple amplifier or wire may not be connected	YES

			Otherwise, if all sensors are connected and can be communicated with a data packet in the form will be returned: $ 1 OK ;	

Load Testing

Table 8: Load Tests

Test Name	Action	Expected Result	Test Passed?
Record Button Spam Testing	Click "Start Recording" button many times in a row in the MacOS application.	Program should output save file dialog once the "Stop Recording" button has been hit and timer stops	YES
Set Flow Button Spam Testing	Click set flow rate numerous times in the MacOS application to test the response of the Arduino and its runtime of commands	Program should continue smoothly and should not freeze or bug out	YES
Minimum Minutes Per Sample Testing	Enter the minimum value into the minutes per sample textbox Click "Start Recording" in the MacOS application	Program UI should not run any slower, and all commands expecting responses sent to the Arduino are responded to	YES
High Arduino Data Query	Initialize test script with correct serial port and with baud rate of 9600 Send numerous polling, setpoint, and temperature request commands per second to the Arduino device	The Arduino should be able to return the expected result (see functional testing) for at least 6 commands / second	YES

Code Quality

In order to ensure code quality and future expandability. All Swift code utilized docC a documentation markup engine for documenting the code for future projects. This includes comments for each class, and major method. Methods should have descriptions of their parameters, function action description, and a description of the data retu rned. Classes should have descriptions of their major functionality and what the class is responsible for. The Arduino C code includes standard inline comments describing major functionality of methods and the intent for each block of code. Wrapping the project together in a cohesive element is the overall project ReadMe which describes how to build the application from scratch, the wiring of the Arduino, and a full specification of the Arduino command API.

Unit Testing Note

Near the end of the project, it was discovered that the AppController should have been implemented using dependency injection instead of as a Singleton class for the sake of testing. Since this change would have required a

significant amount of code rewriting, it was decided that the unit tests would be replaced with corresponding functional and UI tests. This implementation does not affect the functionality of our program negatively, just the implementation of unit tests.

IX. Project Ethical Considerations

ACM/IEEE Principles Applicable to the Project

The main ethical principles that apply to this project are the appropriate handling and communication of risks that could be created by malfunctions in the system and providing high quality products. Since the system created will handle the control of potentially dangerous equipment, it is important for this to be considered in the development of both the software and hardware systems. Additionally, the risks associated with a malfunction should be clearly communicated with the client and any others who may use the system in the future so that they may take any additional steps that they believe are necessary for assuring safety when using our system and the equipment that our system will be controlling. Finally, we should seek high quality with regards to both the product and the processes in compliance with our responsibilities as computer scientists.

ACM/IEEE Principles Most in Danger of Being Violated

ACM principle 2.6, performing work only in areas of competence, is the most likely principle to be violated as this is the first time that any of our members are working in Swift, the language that our application is written in [2]. Due to the time constraint on this project, it is difficult to learn the language to a decent level of competence before completing the software. One of the consequences of this is that the User Interface may not look as professional as it could if our competency level with Swift were higher. Another principle that is likely to be violated is ACM principle 2.9, designing systems that are robust and usably secure [2]. Again, due to the time constraint of the project, it is challenging to implement functionalities other than the core functionalities. Therefore, it is possible that some safety and security measures that should be implemented are not being implemented. At worst, a malfunction in our system could result in a physical hazard relating to the equipment that our system is managing.

Harm Test

Technology has a way of not always working as planned, but this furnace controller is no more hazardous than the previous option. Working in a laboratory with equipment like furnaces that get up to thousands of degrees can be dangerous, so it is necessary to take all precautions when dealing with this technology. Our option will help to automate data collection and provide a more seamless way to control various components of the furnace. There is inherent danger when dealing with these tools, but our controller is just a way to control these devices easier and more safely. In areas of user feedback, the application does less harm as it can error check user inputs and prevent setting flow rates outside of their safe range. Instead, it can issue an alert and prevent the user action.

Reversibility Test

Trading places in the context of this project means taking the place of Dr.Hildreth and the other researchers which utilize the MacOS application to control the furnace. If we were to utilize this application, we would want user errors to be viewed as user errors, and not confusing UI design choices. To ensure this, we enhanced the usability of the application by adding error message dialogs and restrictions on what components of the UI are accessible depending on if an Arduino is connected to the hosting Mac application. For example, the UI textboxes for changing flow rates are not accessible if the Flow Sensor cannot be communicated with. Another important observation for trading places is that lab environments, especially Dr.Hildreth's, are especially dangerous and minimizing the time spent in that environment

through the ability to automate data collection significantly helps those are going to use the software conduct their research in a safer manner.

Ethical Considerations for a Failed Quality Assurance Plan

The most important quality assurance pieces that have ethical considerations are making sure that setpoints have limits to them and making sure our UI is user friendly. It is possible to set a setpoint on the mass-flow controllers that is outside of the range accepted by the flow controller, so testing that the user cannot input a number outside of the range is essential for safety assurance. The consequences of this could potentially be dangerous, as the materials inside the furnace could act differently based on how much gas is going into the furnace. Making sure that Dr. Hildreth and the lab technicians are happy with the user interface and the user experience will ensure that the application is usable and accessible.

X. Project Completion Status

According to our functional requirements and our definition of done described at the beginning of the project, we have fully completed the furnace controller project for this field session. The only unimplemented features of our project are what we listed as stretch goals, including setting the temperature on the furnace and adding Bluetooth communication between the Arduino and the application. Also, we did not add a header to the saved CSV file. Following is a list of features implemented.

- Arduino
    - Created a working circuit which connects the Arduino to all the desired sensors.
        - Successfully communicated between the Arduino and the Flow Meters by using a TTL to RS232 converter.
        - Successfully communicated between the Arduino and the Thermocouple by using a thermocouple amplifier which could communicate with the Arduino via SPI.
    - Designed and implemented a request-response communication protocol to send commands and data between the Arduino and MacOS application
- MacOS Application
    - Created an Arduino controller to send commands to and receive data from an Arduino connected via USB
    - Designed and programmed a simple GUI to interact with the Arduino
        - Created a view to configure the Arduino communications.
        - Created text fields and buttons which could be used to update the setpoints of the flow meters.
        - Created Text objects which would update according to the temperature and flowrate values read from the sensors.
    - Implemented functionality to record an experiment and collect data on a timer
    - Used DataGraph to graph furnace sensor values versus time over the course of an experiment
    - Wrote all recorded values to a CSV file which can be saved to a location on disk selected by the user

XI. Future Work

More implementations

While we did complete all our functional requirements for this project, there were some stretch goals that our client had in mind, but that our team did not have time to implement. One stretch goal was to add the ability to set the temperature of the furnace, but we were never given the hardware specifications that were needed in order to implement this functionality.

There are some improvements that could be made on the Arduino side, including adding Bluetooth and getting a board with more RAM and more serial communication ports. It is possible that Bluetooth may already be supported on the MacOS application side (ORSSerialPort has a Bluetooth port option), and all that is needed is to update hardware and code to talk through Bluetooth through the Arduino. Bluetooth would greatly improve the safety of using the furnace, as it allows there to be more distance between the lab technician and the furnace. Also, if someone wants to expand the Arduino code or add more sensors to talk to, they may want a board with more capabilities because we pushed the limits of the Arduino UNO in our project.

Future work with this project includes some of the previously mentioned stretch goals to create a furnace recipe creator. With this future work, Dr. Hildreth or others who choose to use this program can automate every part of the furnace's function, allowing the user to create schedules for temperature and gas flow fluctuation for whatever recipes they run in the furnace. Because we have implemented close to all functionality of this program except setting temperature, creating this recipe maker should not take long to implement and could work using user-created downloadable scripts to run in the program. When all these possible new features are implemented, this will produce a safer work environment for all furnace users and allow for true remote-control usage and monitoring of this technology.

XII. Lessons Learned

During this project we learned valuable lessons surrounding working together in a team environment, the importance of clear communication, and the value in subscribing to an engineering workflow process like Agile development. Listed below are some of the key project successes that we all are extremely proud to have accomplished, as well as acknowledgment of some of the areas that we encountered adversity.

Key project successes

- Although Swift was a new language to all of us just a couple of weeks ago, having in-depth past coding experience helped us learn the syntax, quickly develop a user interface, and complete much of the backend coding.
- Spending consistent amounts of time on this project every weekday has allowed us to gauge the work we have done very well and has allowed us to complete what we had planned.
- Near the start of the project, we discovered that the sensors we were working with could communicate data using analog voltages, as was mentioned by our client, but they could also communicate data via RS232 Serial. After presenting this discovery to our client, the entire scope of our project changed to communicating with the sensors using RS232 instead of analog voltages in order to take advantage of the commands and logic that already existed within the sensors. This was an important lesson as it revealed that sometimes a better solution may exist outside of the client's expectations for a project.

Key project shortcomings and solutions

- Trying to create a graphing interface within our UI was a challenge considering there was not any good built-in graphing program in Swift until *very* recently (Swift charts was announced the day our project was due to the client). Utilizing our client as a resource helped immensely to get past this hurdle and create a clean UI with a graph which our client 100% approves of. This taught us a valuable lesson that sometimes it easier to admit you are having issues and reach out for support rather than struggling individually.
- The thermocouple outputs very small voltages which would be unreadable in our current system, so we purchased a MAX31855 Thermocouple Amplifier chip to boost the voltages up to a readable level. This was a shortcoming as it was assumed that this would be the easiest hardware component to get working because it just relied on the Arduino's analog to digital converter (ADC). This was almost a critical mistake as this assumption was incorrect and we needed additional hardware to read this device. The lesson. to be learned was always check your assumptions and make sure you are not relying on false assumptions.

- The client, Dr. Hildreth, gave our team a great deal of freedom in designing the GUI and the sensor communication system. Although this was helpful in that it allowed our team to adjust the project's scope as we learned more about the systems and languages that we were working with, this also made it difficult to finalize certain aspects of the project that Dr. Hildreth wanted implemented. For example, it was only discovered later in the project that Dr. Hildreth did not want any Swift alerts because he found them to be overly intrusive. As such, we had to change how our program responded to unexpected inputs by disabling buttons instead of notifying the user through an alert. Dr. Hildreth had previously mentioned that he did not like a different feature because it was intrusive, which should have indicated to us that he disliked intrusive features. This made us realize the importance of deliberate communication with our client about their preferences as well as the importance of reading into the client's preferences based on the information that they gave us.

Key lessons

- Clients are a great resource for getting assistance
- Take charge of getting something done
- When there is a problem ask questions to get to the solution
- Taking mental breaks is good for getting perspective to solve an issue

XIII. Team Profile

Nathan Bujarski

Year: Senior
Studying: Computer Science
Hometown: Highlands Ranch, CO
Bio: Nathan enjoys reading books, playing videogames, and learning martial arts.

Josh Mendelsohn

Year: Junior
Studying: Computer Science major and Engineering Physics minor
Hometown: Baltimore, MD
Bio: Josh loves rock climbing, backpacking, and astronomy.

Colton Meyers

Year: Graduating Senior
Studying: Computer Science
Hometown: Conifer, CO.
Bio: Colton enjoys programming, skiing, and rock climbing.

Preston Yates

Year: Senior
Studying: Computer Science focus in Data Science
Hometown: San Antonio, TX
Bio: Preston enjoys spending time with friends, gaming, and learning about Rubik's cubes

References

[1] O. Hildreth, "Hildreth Research Group," *Colorado School of Mines*. [Online]. Available: https://hildrethlab.mines.edu/. [Accessed: 13-Jun-2022]

[2] "ACM Code of Ethics and Professional Conduct," *ACM*. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 17-Jun-2022]

[3] "IEEE Code of Ethics," *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 17-Jun-2022]

[4] "What is Arduino?," *Arduino*. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed: 19-May-2022]

[5] "Serial communication Basic Knowledge -RS-232C/RS-422/RS-485," *CONTEC*. [Online]. Available: https://www.contec.com/support/basic-knowledge/daq-control/serial-communicatin/. [Accessed: 19-May-2022]

[6] "Serial Communication Methods – Synchronous & Asynchronous," *PIJA Education*, 15-Jul-2021. [Online]. Available: https://pijaeducation.com/communication/serial-communication-methods-synchronous-asynchronous/. [Accessed: 10-Jun-2022]

[7] "4.2 Synchronous Serial Communication," *Learn about the Serial Peripheral Interface (SPI) · VectorNav*. [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-primer/hardware/synccomm#:~:text=One%20of%20the%20most%20common,bit%20rates%20exceeding%2010%20MHz. [Accessed: 10-Jun-2022]

[8] "4.1 Asynchronous Serial Communication," *VectorNav*. [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-primer/hardware/asynccomm#:~:text=Asynchronous%20serial%20communication%20is%20a,end%20of%20a%20data%20message. [Accessed: 07-Jun-2022]

[9] S. Campbell, "Basics of UART Communication," *Circuit Basics*, 14-Nov-2021. [Online]. Available: https://www.circuitbasics.com/basics-uart-communication/#:~:text=UART%20stands%20for%20Universal%20Asynchronous,transmit%20and%20receive%20serial%20data. [Accessed: 10-Jun-2022]

[10] T. Sharma, "RS232 Serial Communication Protocol: Basics, Working & Specifications," *What is RS232 Serial Communication Protocol? RS232 Basics, Working & Specifications*, 01-Jan-2018. [Online]. Available: https://circuitdigest.com/article/rs232-serial-communication-protocol-basics-specifications. [Accessed: 19-May-2022]

[11] jimblom, "RS-232 vs. TTL Serial Communication," *SparkFun*, 23-Oct-2010. [Online]. Available: https://www.sparkfun.com/tutorials/215#:~:text=This%20method%20of%20serial%20communication,'0')%20is%200V. [Accessed: 19-May-2022]

[12] Anusha, "Basics of serial peripheral interface (SPI)," *Electronics Hub*, 06-Oct-2021. [Online]. Available: https://www.electronicshub.org/basics-serial-peripheral-interface-spi/#:~:text=Serial%20Peripheral%20Interface%20or%20SPI,a%20microcontroller%20and%20its%20peripherals. [Accessed: 07-Jun-2022]

[13] "VCC and VSS Pins," *tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/vcc-and-vss-pins#:~:text=VCC%20(Voltage%20Common%20Collector)%20is,Supply)%20means%20ground%20or%20zero. [Accessed: 19-May-2022]

[14] “Swift,” *Apple Developer*. [Online]. Available: https://developer.apple.com/swift/. [Accessed: 19-May-2022]

[15] P. Hudson, “What is SwiftUI?,” *Hacking with Swift*, 09-Feb-2021. [Online]. Available: https://www.hackingwithswift.com/quick-start/swiftui/what-is-swiftui. [Accessed: 10-Jun-2022]

[16] “DocC,” *Apple Developer Documentation*. [Online]. Available: https://developer.apple.com/documentation/docc. [Accessed: 10-Jun-2022]

[17] The Editors of Encyclopaedia Britannica, “Thermocouple,” *Encyclopædia Britannica*. [Online]. Available: https://www.britannica.com/technology/thermocouple. [Accessed: 19-May-2022]

[18] R. Awati, “What is the Seebeck effect?” *SearchNetworking*, 18-Oct-2021. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/Seebeck-effect#:~:text=The%20Seebeck%20effect%20is%20a,difference%20between%20the%20two%20substances. [Accessed: 16-Jun-2022]

[19] “Operating Manual For Mass Flow Controllers Models MC · MCD · MCE · MCQ · MCR · MCS · MCV · MCW,” *Alicat Scientific*. [Online]. Available: https://documents.alicat.com/manuals/DOC-MANUAL-9V-MC.pdf. [Accessed: 19-May-2022]

[20] “Operating Manual For Mass Flow Meters Models M · MQ · MS · MW · MB · MBQ · MBS · MWB,” *Apex Vacuum*. [Online]. Available: https://apexvacuum.com/wp-content/uploads/2021/09/DOC-MANUAL-M-2021-Apex.pdf . [Accessed: 19-May-2022]

[21] “Arduino® Uno R3,” *Arduino*. [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf. [Accessed: 17-Jun-2022]

[22] “SP3222EB/3232EB True +3.0V to +5.5v RS-232 Tranceivers,” *SparkFun*. [Online]. Available: http://cdn.sparkfun.com/datasheets/Components/General%20IC/SP3232EBCA-L.pdf. [Accessed: 17-Jun-2022]

[23] “MAX31855 Cold-Junction Compensated Thermocouple-to-Digital Converter.” [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf. [Accessed: 17-Jun-2022]

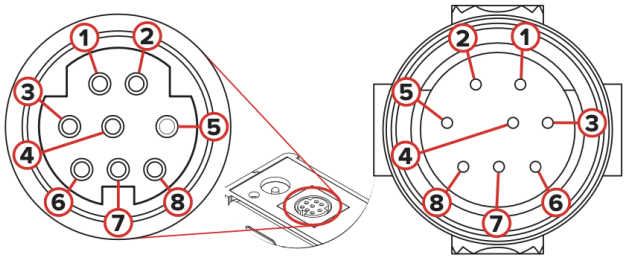
Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Arduino	"...an open-source electronics platform based on easy-to-use hardware and software" [4].
Mass Flow Meter	A device which measures the volumetric and mass flow rate of a gas flowing through the sensor
Serial Communication	A method of communication between devices which involves sending and receiving one bit at a time at a specified baud rate [5].
Synchronous Serial Communication	Serial data signals are synchronized using a common clock signal. Faster than asynchronous serial communication but requires more wires [7].
Asynchronous Serial Communication	Serial data signals are not synchronized using a common clock signal and instead use start and stop bits to indicate the completed transfer of a data package [8].
UART	A device which can commonly be found in microcontrollers and other electronic devices that handles serial communication between devices. UART stands for Universal Asynchronous Receiver/Transmitter [9].
RS-232 Serial	A serial communication protocol that can be used over medium distances and can handle both synchronous and asynchronous data signals. RS-232 stands for "Recommended Standard 232." In RS-232, a 1 is represented as a negative voltage and a 0 is represented as a positive voltage [10].
TTL Serial	A type of asynchronous serial communication which is used to transmit data over short distances. TTL stands for "transistor-transistor logic." In TTL, a 1 is represented using the VCC and 0 is represented with 0V [11].
SPI Serial	A type of synchronous serial communication that uses a master-slave configuration. Requires four different signals: MISO, MOSI, Serial Clock, and Chip Select. SPI stands for "Serial Peripheral Interface" [12].
VCC	"VCC (Voltage Common Collector) is the higher voltage with respect to GND (ground)" [13]. VCC is usually either 3.3V or 5V.
XCode	A development environment to build MacOS and IOS applications using Swift, SwiftUI, and Objective-C

Swift	A general-purpose compiled programming language for iOS, iPadOS, macOS, tvOS, and watchOS [14].
SwiftUI	"...a user interface toolkit that lets us design apps in a declarative way" [15].
DocC	A documentation compiler created by Apple to turn markdown comments in documentation for Swift and Objective-C projects [16].
Thermocouple	A thermocouple is an electrical device consisting of two dissimilar electrical conductors forming an electrical junction. A thermocouple produces a temperature-dependent voltage as a result of the Seebeck effect, and this voltage can be interpreted to measure temperature [17].
Seebeck effect	The Seebeck effect is a phenomenon in which a temperature difference between two dissimilar electrical conductors or semiconductors produces a voltage difference between the two substances [18].

Appendix B – Additional Design Documents



Female Connector: Device Male Connector: Cable

Standard 8-Pin Mini-DIN Pinout

Pin	Function	Cable color
1	Not Connected (or optional 4–20 mA Primary Output Signal)	Black
2	Static 5.12 Vdc by default. Optional: Secondary Analog Output (4–20 mA, 0–5 Vdc, 1–5V dc, 0–10 Vdc) or Basic Alarm	Brown
3	Serial RS-232 RX / RS-485(–) Input Signal (receive)	Red
4	Analog Setpoint Input	Orange
5	Serial RS-232 TX / RS-485(+) Output Signal (send)	Yellow
6	0–5 Vdc (or optional 1–5 Vdc or 0–10 Vdc) Output Signal	Green
7	Power In (as described above)	Blue
8	Ground (common for power, digital communications, analog signals and alarms)	Purple

Appendix Figure 1: Mini-DIN pinout

Appendix C – Parts Used

Item Name:	Amount:	Data Sheet:	Part Number:
ELEGOO UNO R3 Board ATmega328P	1	https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf	EL-CB-001
Breadboard	1	N/A	N/A
RS232 to TTL Converter	1	http://cdn.sparkfun.com/datasheets/Components/General%20IC/SP3232EBCA-L.pdf	SP3232EB
MAX31855 (Thermocouple Amplifier)	1	https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf	MAX31855K
USB to USB-C Dongle		N/A	N/A