Jumping Jacks

Final Report and User's Manual
April 16, 2021

Kristin Hamman: kah635@nau.edu
Richard Hutchinson: rdh258@nau.edu
Yuhau Wei: yw249@nau.edu
Elizabeth Zyriek: eaz34@nau.edu

EE486C
Team 4
Gas Reactor Sensor
April 16, 2021

Table of Contents

3. Introduction

a. Background of your client

The client for the Gas Reactor Project is Christopher Ebert and the Schuur Lab. The Schuur Lab is a research lab focused on the interactions between terrestrial ecosystems and global exchange, the exchange of carbon plants, soils, and the atmosphere, understanding the response of terrestrial ecosystems to changes in climate and disturbance regimes, responses of arctic ecosystems to climate change, and radiocarbon dating. The project focuses on making the Isotope Sample Preparation Lab more power efficient and up to date in terms of electronics. The project outline was to enhance, automate, and make a pressure and temperature monitoring system that the client uses to work on carbon samples to create needed isotopes. The client expressed issues of the old system which were: old parts being discontinued, old power supplies coming to an end, and the need for manual control throughout the process. The automation of the sampling process would help the client indefinitely as the reaction can take anywhere from 2-5 hours and has to be manually turned off using the current system. With the proposed solution this process would happen on its own after the pressure starts to level out for a prolonged period of time. Although, having a pressure and temperature system work together is not uncommon; the niche aspect comes in with the magnitude the pressure and temperature systems have to be able to read and reach. The carbon samples will have to be heated up anywhere from 600 to 900 degrees Celsius. The need for this is due to carbon's ability to withstand extreme temperatures and turn into three usable isotopes for analysis. With the ever growing issue of climate change the balance of carbon becomes more important because the imbalance of carbon on earth and in the atmosphere can cause issues such as global warming and climate disruption. This is the main reason the client needs to come up with a new system before the old one is not usable anymore. The Schuur Lab’s Assistant Director, Mr. Ebert, works on these carbon isotope samples to use for analysis after the reactions occur in small test tubes.

b. The problem being solved

The focus of this project is to create a new set of ovens to work in the Schuur Lab. The pieces used in the current set are out of date and require updating, so this new system is built around keeping the Schuur lab running without the need to replace equipment. This project will be used to help monitor and change temperatures for different carbon reactions, where the user can observe and directly change the temperature of the different ovens in the lab. We are updating the heaters used in the lab to increase the temperature

of a carbon reaction by using a proportional (P), integral (I), derivative (D) (PID) temperature controller that works in conjunction with a pressure system. The project would be able to read the pressure of 13 samples, 12 arranged by the client and a reference pressure. The project then uses the PID temperature controller to reach a desired “setpoint” chosen by the user and left alone while the reactions go to completion. The goal is to have an Arduino microcontroller get inputs from the pressure and heating sensors attached to the vacuum chamber where the reaction occurs. The values from these sensors will then be displayed on a Liquid-Crystal Display (LCD) screen. Our sensors will be used for different aspects, such as temperature measurement, precision adjustment, information transmission, and data display. A user input will be needed to set the temperature, which is also displayed on the LCD screen. The pressure sensor will be used to monitor the reaction and determine whether the heating block should be on or off based on its readings. Thus, this Gas Reactor Project will allow for samples to be properly made and tracked by our PID temperature controller and pressure system.

4. Design Process

A. Overall design process description

The following Figure 1 shows the system architecture at Level 1. The important requirements that our team identified for this project in relation to the architecture are listed below.

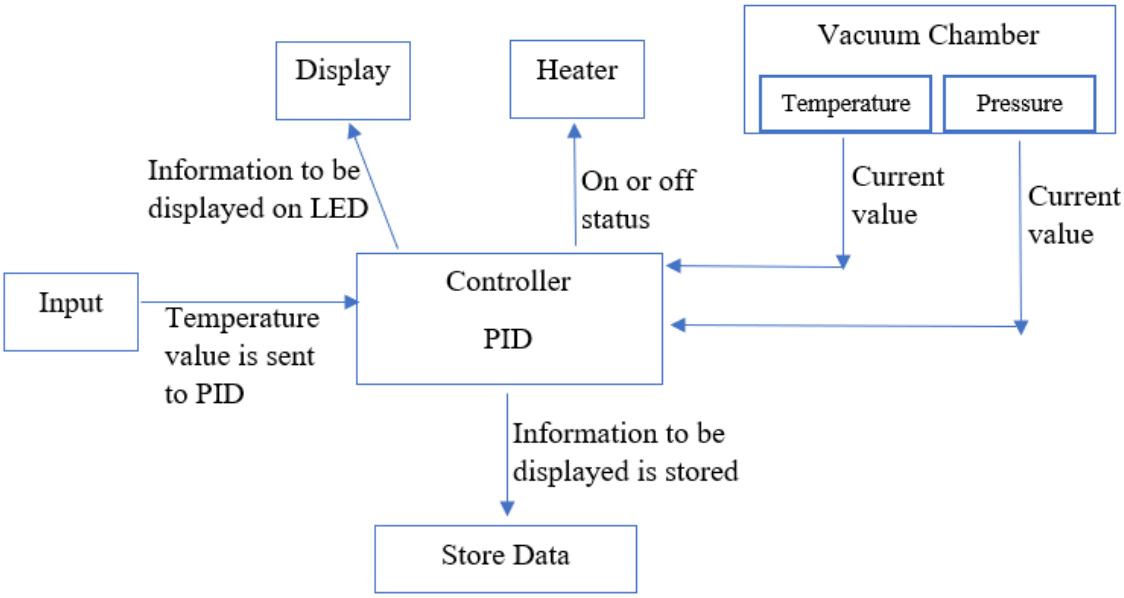


Figure 1: System Architecture- Level 1

- The Arduino microcontroller will get inputs from the pressure and heating sensor that are attached to the vacuum chamber where the reaction occurs.
- The values from the pressure and temperature sensor are then displayed on an LCD screen for easy viewing.
- A user input will be needed for the “set temperature” which is also displayed on the LCD screen.
- The pressure sensor is used to monitor the reaction and determine whether the heating block should be on or off based on its readings.
- Store the stop time of the reaction and print it on the LCD screen.

B. Functional decomposition

The following figures and explanations detail the Subsystem Architecture for Level 2.

The first subsystem is for the input. In this subsystem, the user will set the temperature between 550-950 degrees Celsius with push buttons. The set temperature value will then be sent to the PID Controller.

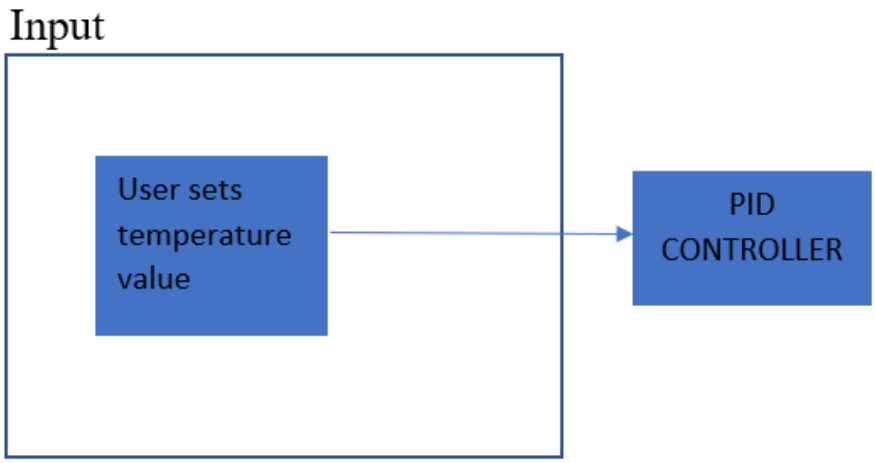


Figure 2: System Architecture (Level 2) for Input

The second subsystem is for the display. In this subsystem, the PID Controller will have the actual temperature, set temperature, and pressure values and the stop time. This information will be sent to the LCD screen to be displayed.

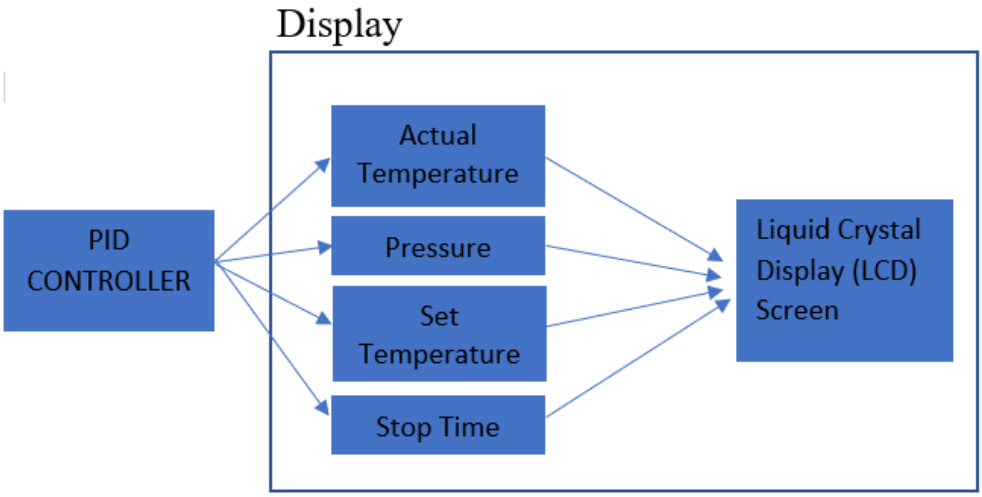


Figure 3: System Architecture (Level 2) for the Display

The third subsystem is for the PID controller. In this subsystem, the user input as well as the information from the vacuum chamber will be sent to the Arduino Uno. The Arduino will then send information to the display, tell the heater if it needs to adjust.

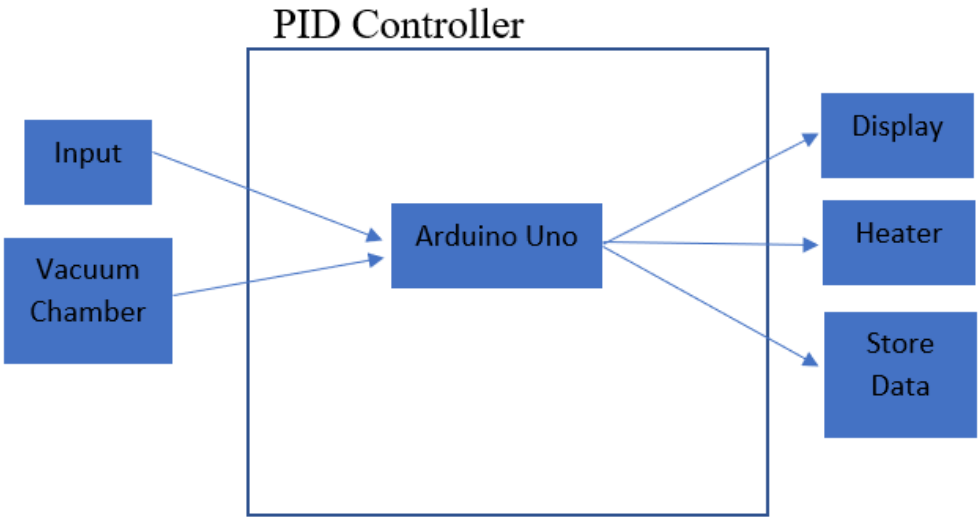


Figure 4: System Architecture (Level 2) for PID Controller

The fourth subsystem is for storing data. In this subsystem, the PID Controller code determines which information needs to be saved i.e temperature, pressure, and stop time. Some of the saved information will be sent to the Display later on.

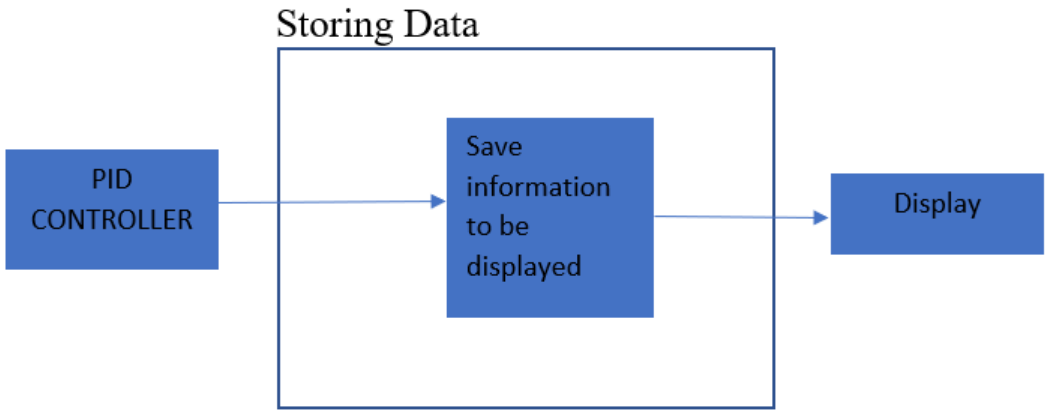


Figure 5: System Architecture (Level 2) for Storing Data

The last subsystem is for the vacuum chamber. The vacuum chamber is already created in the lab with the reaction occuring inside. The pressure sensor is attached to the vacuum chamber while the temperature sensor is integrated into the k-Type thermocouple. Information is then sent to the PID Controller.

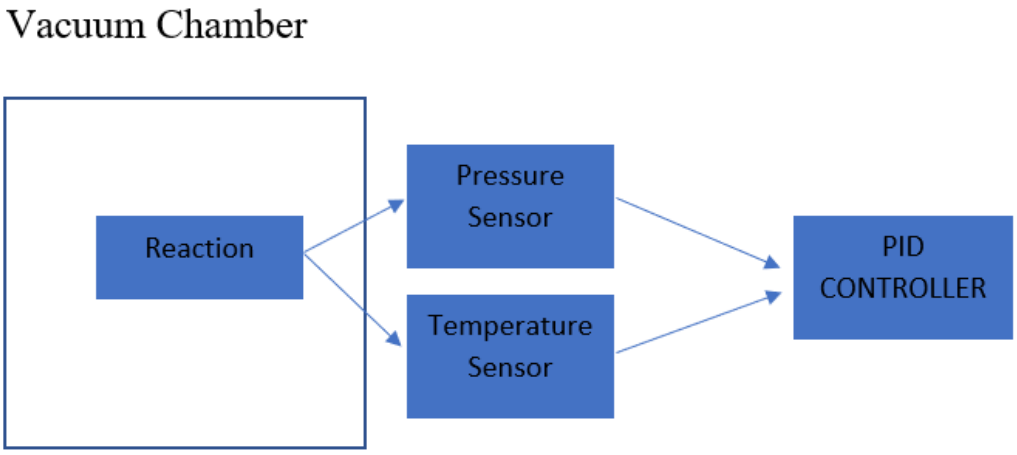


Figure 6: System Architecture (Level 2) for the Vacuum Chamber

C. Prototype findings: results or effect

This capstone project is the Gas Reactor Sensors. The client, Chris Ebert, is the manager of the Ted Schuur lab on the Northern Arizona University Campus. He works in the sample preparation lab where the gas reaction systems operate. This team has been asked to assist in updating the heaters used in the lab to increase the temperature of a carbon reaction. The goal is to have an Arduino microcontroller get inputs from the pressure and heating sensors attached to the vacuum chamber where the reaction occurs. The values from these sensors will then be displayed on a Liquid-Crystal Display (LCD) screen. A user will input the temperature, which is also displayed on the LCD screen. The pressure sensor will be used to monitor the reaction and determine whether the heating block should be on or turned off based on its readings.

For the first prototype, Elizabeth worked with the LCD screen. The LCD must receive information from the proportional integral derivative (PID) controller and then display it. The prototype is the part of the system architecture that directly connects the PID controller to the display by sending the information to be displayed. This prototype will reduce the risk of me using an LCD display for the first time. By figuring out how to program the board with simple code, the team can later write more extensive code that

involves the PID sending the temperature information. This will save time later when building the project.

The biggest challenge to making the prototype was writing the code. At first, all of the information, the “set temp:,” “real temp:,” and the values were being displayed at the same time because of how the loop was written. Two delays needed to be included, one after each header and temperature value so that the information would be displayed correctly. The information is displayed for two seconds on the LCD screen before changing. The prototype did not take as long as expected to complete. The coding took about an hour to write by looking at various Arduino sources online. The wiring of the board also took about an hour since it went slowly and carefully to avoid making any mistakes. After this was completed, the code was able to compile and upload to the Arduino. The program was then run to verify that the screen was working. The results from this prototype will influence the project. The team can be told that the LCD screen is usable, and utilizing the one that the previous capstone team bought would work successfully. There is no need to find a new type of display, purchase it, or figure out how it works.

The second prototype, built by Kristin, was part of the code used to allow push buttons to change the temperature of the vacuum. Two buttons were used to increment and decrement user input for temperature by 10 degrees. Although it is currently not communicating with the temperature device, this is the first step to that happening. This data was given to other team members to take the data and display it on an LCD in real time to show the current working temperature from the user.

This prototype will fit into the input aspect of the system architecture. It connects user input to the Controller PID in order to change the temperature of the Vacuum Chamber. From this prototype she hoped to learn more about what our Arduino will be functioning as. It will also give a better understanding as to how the Controller PID is able to communicate with the vacuum to actually alter the temperature that it is operating at. The initial code for using buttons to operate on an Arduino was easier to understand but finding how it integrates with the Arduino and its individual parts was more difficult. The manual operation of temperature is a pivotal aspect of the design, as a specific operating temperature for the system is required. If this part of the process does not work properly then an important requirements specification for the project is lost. By ensuring that this part of the project works properly, this requirement can also be fulfilled.

The main principle of the next prototype is: the voltage that the sensor outputs changes accordingly to the gas level that exists in the atmosphere. The sensor outputs a voltage

that is proportional to the concentration of gas. In other words, the relationship between voltage and gas concentration is the following: the greater the gas concentration, the greater the output voltage the lower the gas concentration, the lower the output voltage. Thus, codes are used to calculate the concentration with the voltage data.

This prototype uses the UNO starter kit to create a gas sensor mode which can detect the concentration of gas nearby. From this, how to design a physical model from some drafts can be better understood. The biggest challenge from this was to choose suitable values of some components by calculation, like the resistor, to fit with the requirement voltage. This work can reduce the project risk by providing a primary model to base design requirements upon.

In the demonstration for this prototype, there was a failure to show the complete prototype. Because the prototype was quickly assembled, but the calculations were not finished it did not work. The biggest challenge faced was working on the prototype in China, with the rest of the teammates in the U.S. There was an overall lack of communication. Regarding the calculation, there was simply no time to complete that aspect of the project. However, for the design draft, it took a lot of time to figure out to convert the prototype from design to board. This prototype is the main part of the hardware design of the project. Even though there were some defects, it provided a primary model for our project.

The physical prototype piece that was built by Richard was the liquid crystal display (LCD) and push-buttons. He had the opportunity to wire everything as well as program both components of the system. He had never worked with a 16 output LCD screen before this and it was very enlightening. Richard had a couple of challenges along the way however, he overcame those and got the circuit to work with code written by Kristin, Elizabeth, and Yuhao.

The prototype is the main essential piece of the project. Without having a display and a way to change the desired “setpoint” the project is useless. That is why he wired the entirety of the Liquid-Crystal-Display (LCD) screen and push buttons as well as programmed some of it. By doing the prototype he learned how to utilize a non-inter-integrated (I2C) LCD screen and the troubles that come with doing so. he expected the programming to be a lot more difficult than the wiring of the board, however, when it came down to it, it was quite the opposite. Although, he would have liked to prototype the functionality of the heater. The parts to regulate this did not come in. This prototype reduces the risk of accidents to the main heater. When the setpoint is in place the real temperature will follow and the only things that need to be changed are the

proportional (P), integral (I), and derivative (D) variables which can be done quickly by changing the associated number in the Arduino code.

The biggest challenge in making the prototype function was not having a potentiometer. There were no videos on how to hook up a 16 pin LCD without the use of a potentiometer. After many hours searching through forums and analyzing the code of other people with the same board there was a simple solution that fixed the problem. If you do not want to attach a potentiometer there needs to be an "analogWrite(digitalPin#, Value)" line that gives the background a brightness value or it will not work. This prototype took longer than expected because of a lack of knowledge of non-I2C LCDs and contrast voltage.

The demonstration of the LCD and push buttons worked to control and display the setpoint on the screen. The code correctly adjusted the setpoints by adding or subtracting a value of ten while simultaneously updating the screen. This was the exact result that was expected out of the prototype. The figures below show the output of the Arduino on startup and when the push-button to increase the setpoint is pressed 5 times. It was a successful demonstration because our hardware worked with the code as we thought it would. Because it was successful, the results were what we expected them to because. Overall, the different prototypes that we each had that were put together worked as we wanted them to.

5. Final Design

a. System architecture with supporting details such as behavior, flowcharts, schematics, diagrams, etc.

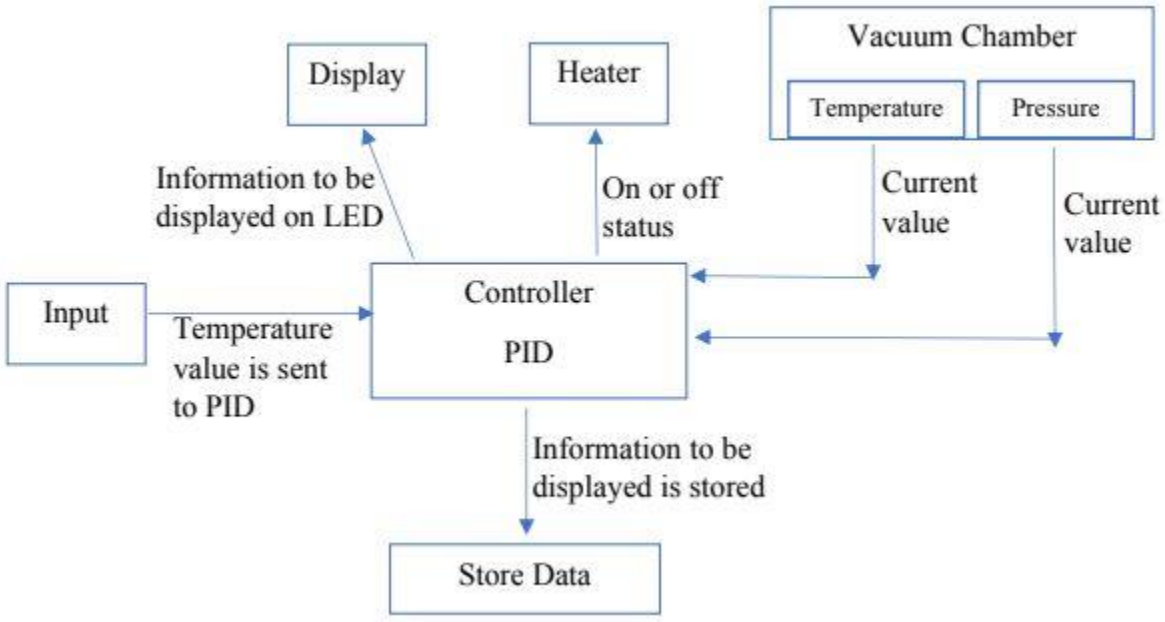


Figure 7: System Architecture

The above diagram shows an overview of the system architecture. This is the basic structure of the system. The below figures each show detail into different components or parts of it; input, storing data, vacuum chamber, display, and controller PID.

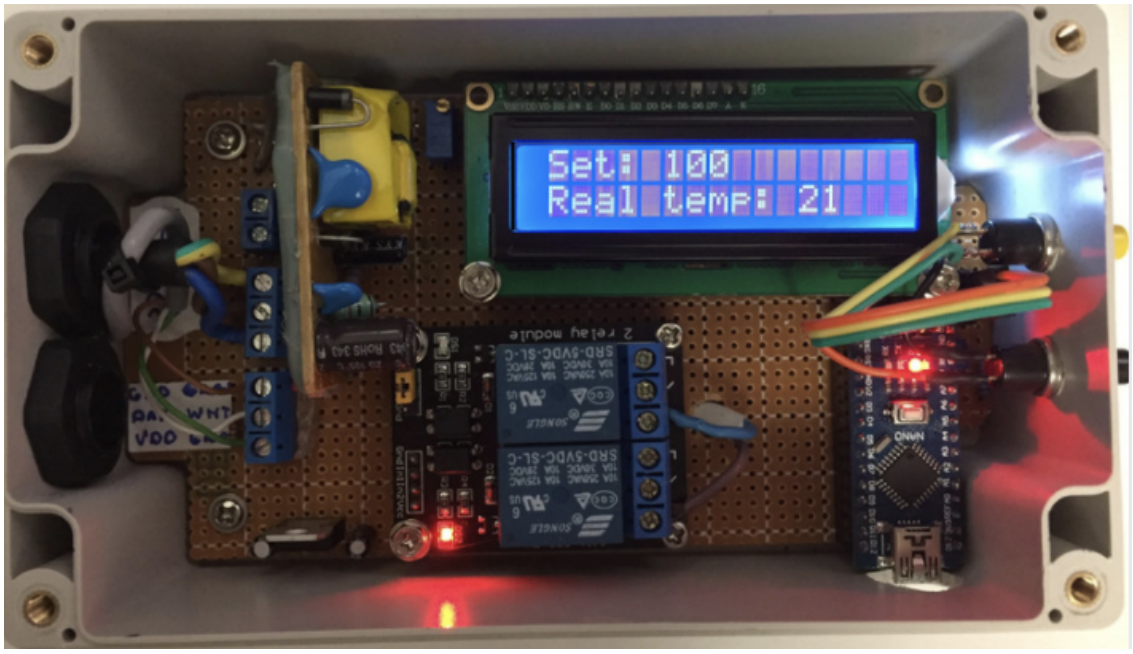


Figure 8: Inside of Our PID Temperature Controller

The above diagram shows an inside look of our PID Temperature Controller.

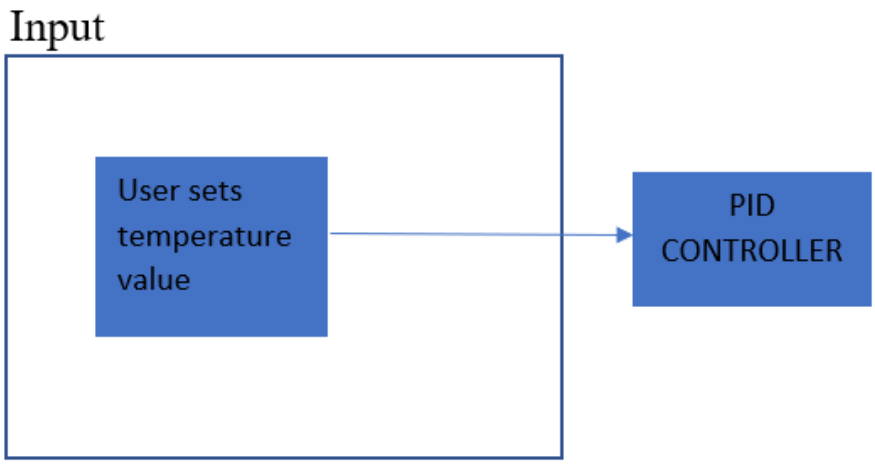


Figure 9: System Architecture (Level 2) for Input

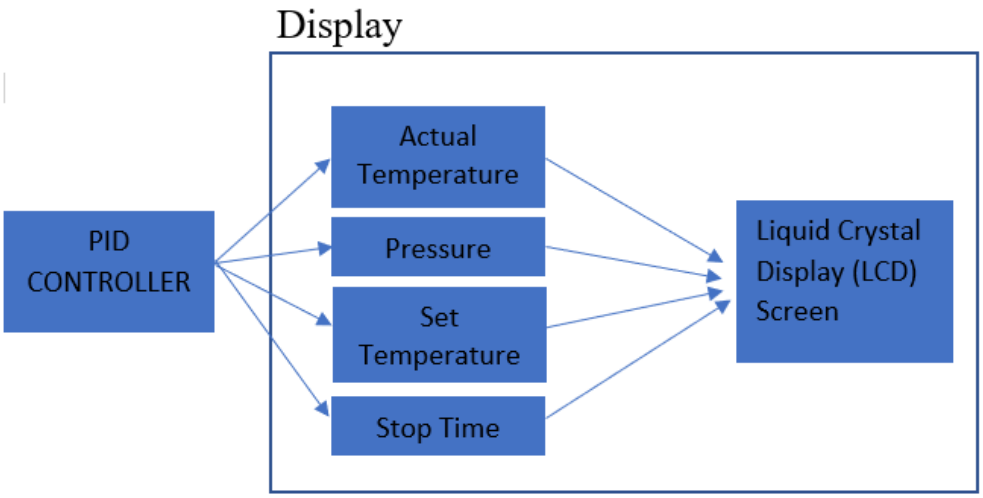


Figure 10: System Architecture (Level 2) for the Display

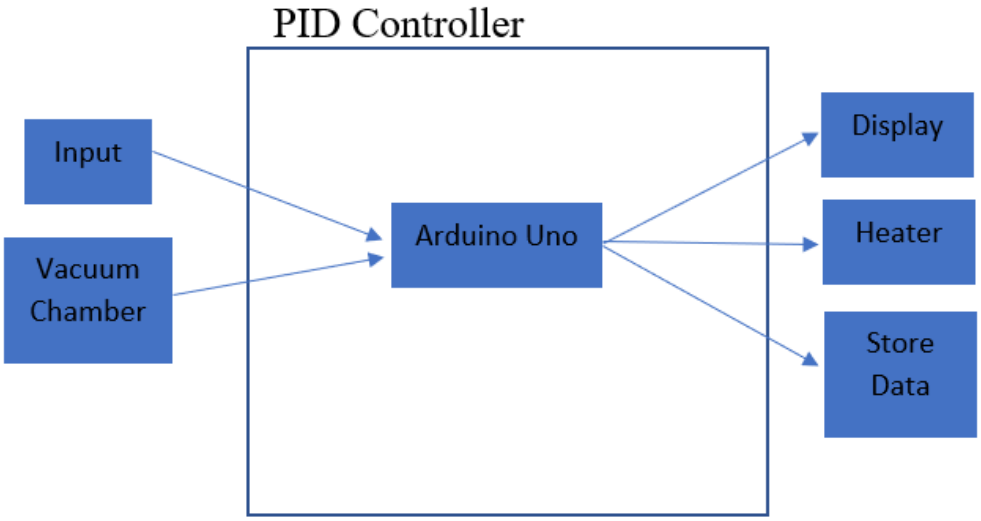


Figure 11: System Architecture (Level 2) for PID Controller

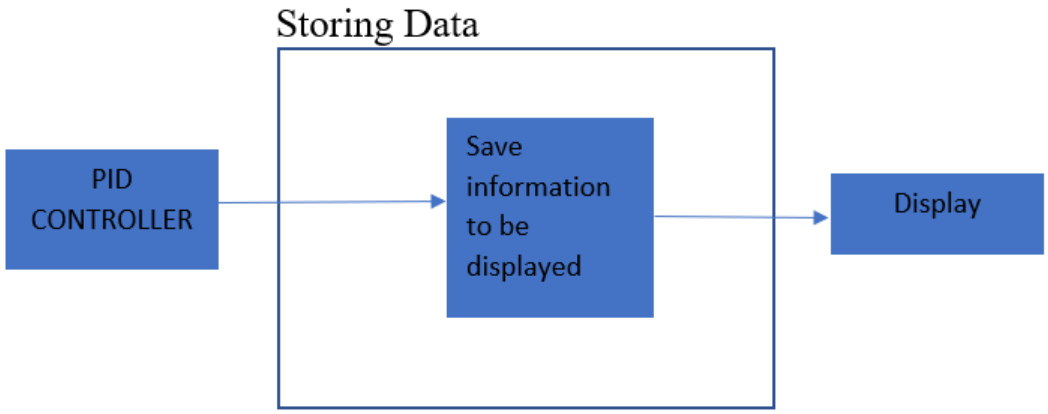


Figure 12: System Architecture (Level 2) for Storing Data

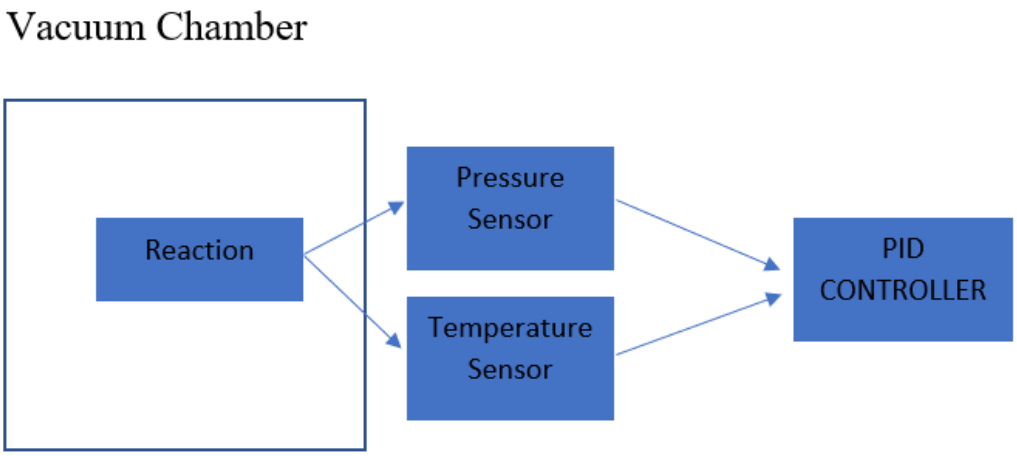


Figure 13: System Architecture (Level 2) for the Vacuum Chamber

b. Text explanations of the major components of your system

There are six main components to our system. The first, the input, is where the user will set a temperature between 550-950 degrees Celsius with push buttons, where it will then be displayed on an LCD. The set temperature value will then be sent to the PID Controller.

The second major component is the display, which is the LCD screen. In this component, the PID Controller will have the actual temperature, set temperature, and pressure values

and the stop time. This information will be sent to the LCD screen to be displayed.

The third main component is the PID controller. In this component, the user input and information from the vacuum chamber will be sent to the Arduino Uno. The Arduino will then send information to the display, tell the heater if it needs to adjust.

The fourth main component is the storage of data. In this component the PID Controller code determines which information needs to be saved i.e temperature, pressure, and stop time. Some of the saved information will be sent to the Display later on.

The fifth major component is the vacuum chamber. The vacuum chamber is already created in the lab with the reaction occuring inside. The pressure sensor is attached to the vacuum chamber while the temperature sensor is integrated into the k-Type thermocouple. Information is then sent to the PID Controller.

6. Results

a. The requirements spreadsheet with color indication of requirements testing results

	A	B	C	D
1			Instructions: List all of your requirements, and use a numbering system.	
2				
3	Type of Test	Status	Req #	Requirement
4				
5	Integrate	*	1	LCD Screen
6	UTM	*	1.1	Temperature
7	UTM		1.2	Pressure
8	UTM		1.3	Time of reaction to nearest second
9	UTM		1.4	Heater on/off status
10			2	Input
11	UTS		2.1	User selects temperature between 300-400 degrees Celsius in 10 degree steps
12			3	Performance of PID controller
13	UTS		3.1	Monitor the reaction
14	UTS		3.2	Check pressure value to decide if heater should be on or off
15			4	Store Data
16	Integrate		4.1	Information that needs to be sent to the Display
17	UTS		4.1.1	Temperature
18	UTS		4.1.2	Pressure
19	UTS		4.1.3	Time of reaction to nearest second
20	UTS		4.1.4	Heater on/off status
21			5	Vacuum Chamber
22	UTS	*	5.1	Temperature
23	Inspect		5.1.1	Current value of temperature will be sent to controller every 0.5 - 5 seconds
24	UTS		5.2	Pressure
25	Inspect		5.2.1	Current value of pressure will be sent to controller every 0.5 - 5 seconds
26			6	Heater
27	UTS	*	6.1	Status of either on or off
28				

Figure 14: Requirements Spreadsheet with Testing Results

b. Important test results

There are four important tests that we used. The first is the **Unit Matrix Test**. We chose to use this test to determine the functionality of our PID, ensuring that the input temperature will be the temperature that the oven goes to. The second test is the **Unit Step-By-Step Test**. We used this in two places, to test for Arduino to Arduino communication and to ensure that our temperature sensor was working correctly. The third test is an **Integration Test**, which checks that the major modules of the overall

system operate correctly together. We used this test to make sure all of the correct data we need displays on the LCD Screen. The last test is the **Inspection Test**, which is done by simply looking at the system. We used this test to make sure that the current value of the temperature displays and is sent to the controller every 0.5 to 5 seconds.

Unfortunately, the results were not as expected. We never received the full power values from the power supply mini-circuit because we could not figure out how to control the voltage and current accurately enough to get consistent temperatures using the PID controller. We failed the integration test and partially failed the Arduino to Arduino communication due to Arduino limitations discovered when testing this aspect. We found out that the wired connections between Arduinos and the LCD screen utilize the same SDA and SCL input and output pins which only allows for one device to be wired and communicate effectively between the two devices.

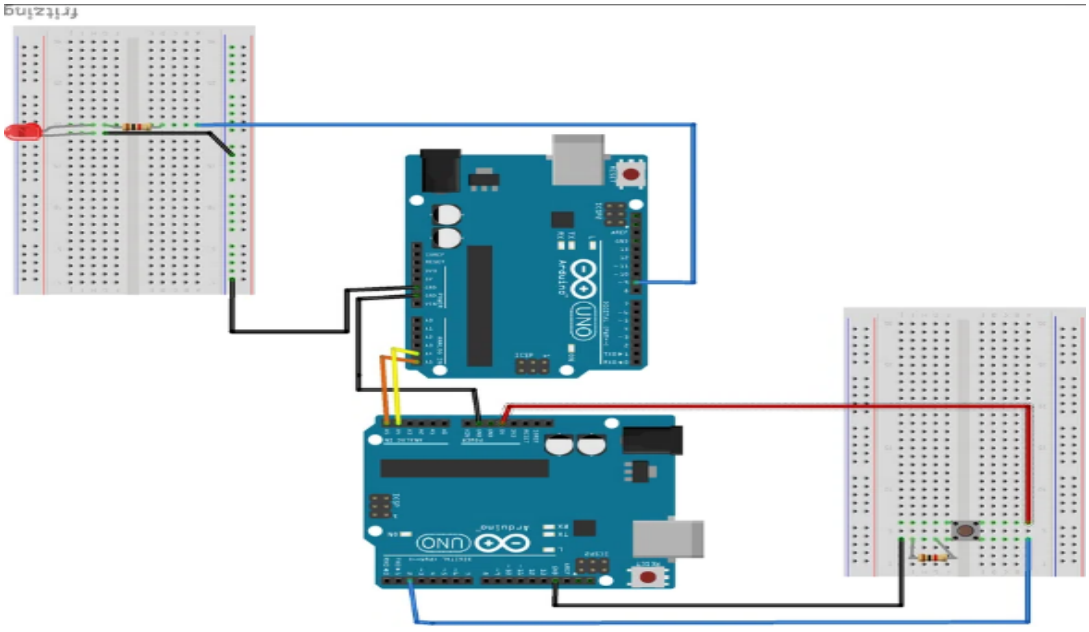


Figure 15: Arduino to Arduino Communication Testing Setup

c. Analysis of Results

In the testing of our project, we encountered a few roadblocks that were described in the Major Tests and Analysis of Results sections of this document. Due to issues we faced while testing, we had to change and get rid of some of our requirements to complete the project. Also, during testing we had to modify different aspects of our design. We initially were working with an I2C LCD screen but had to change to a regular LCD screen when we faced issues with the first one. After we began testing, we needed more space on our

Arduino Board so had to change back to the I2C LCD screen to save space on the board for other parts of the project. Although we modified our design, we still did not have space on our board to operate the LCD screen and use Arduino to Arduino communication as they required the same pins. As this was done at the end of our testing, we did not find a quick and easy solution. In the future we could have experimented with different forms of Arduino communication that were not wired.

We did a number of regression tests with our design, as the project was dependent on the code working properly with our hardware. This is how we were able to fix bugs that came up as we were completing our other testing. For example, this was done when we encountered issues with our LCD screen when we were testing our Arduino to Arduino communication. The LCD screen worked before we set up the communication but would not as we were testing the functionality of the communication. As we were testing, the importance of testing procedures became apparent. There are aspects of the project that we would not have known to check on until working with the final product if we had not tested them, the main example being our communication system working with the LCD screen. Testing was important to discover the issue and, with more time, find a different approach.

7. Conclusion of Capstone Report

a. Most important requirements and their results

The first major test that was done was the verification of the temperature system and the K-type thermocouple that is used to read high temperatures. This test was done to verify the accuracy of not only the temperature sensor itself but help the team find out what adjustments needed to be made in the PID controller manual. For this test it was clear the K-type thermocouple was not the most accurate device but when researched and discussed with the client this was found to be normal and the client gave us the green light to continue with this thermocouple. In the chart below it shows the accuracy of the max6675 temperature sensor and thermocouple compared to a laser temperature sensor. The chart plots the difference between the laser temperature reader and the temperature sensor (k-type thermocouple) temperatures.

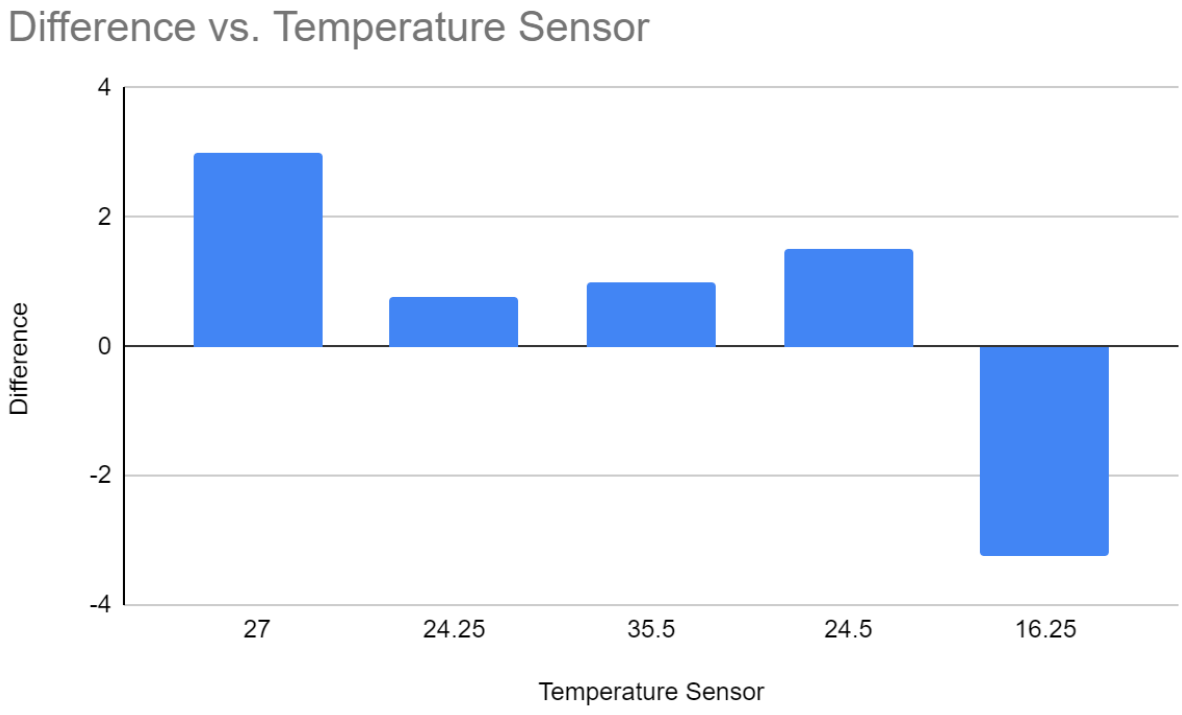


Figure 16: Difference in Temperature Sensor and real temperature

As you can see in the chart it is inconsistent in its ability to detect temperature at times. The chart shows massive differences while being cooled and slightly heated.

The second major test was the Arduino to Arduino communication between the pressure and temperature systems. In this test we did a number of sub-tests from sending a number from one Arduino to another and printing it out to the serial monitor to trying to display the received number or character to a liquid crystal display (LCD) screen. The image below shows the characters being sent from Arduino master to Arduino slave. The code below has the master send a value of x to the slave arduino which sends back the word "hello" every time the "x" value changes.

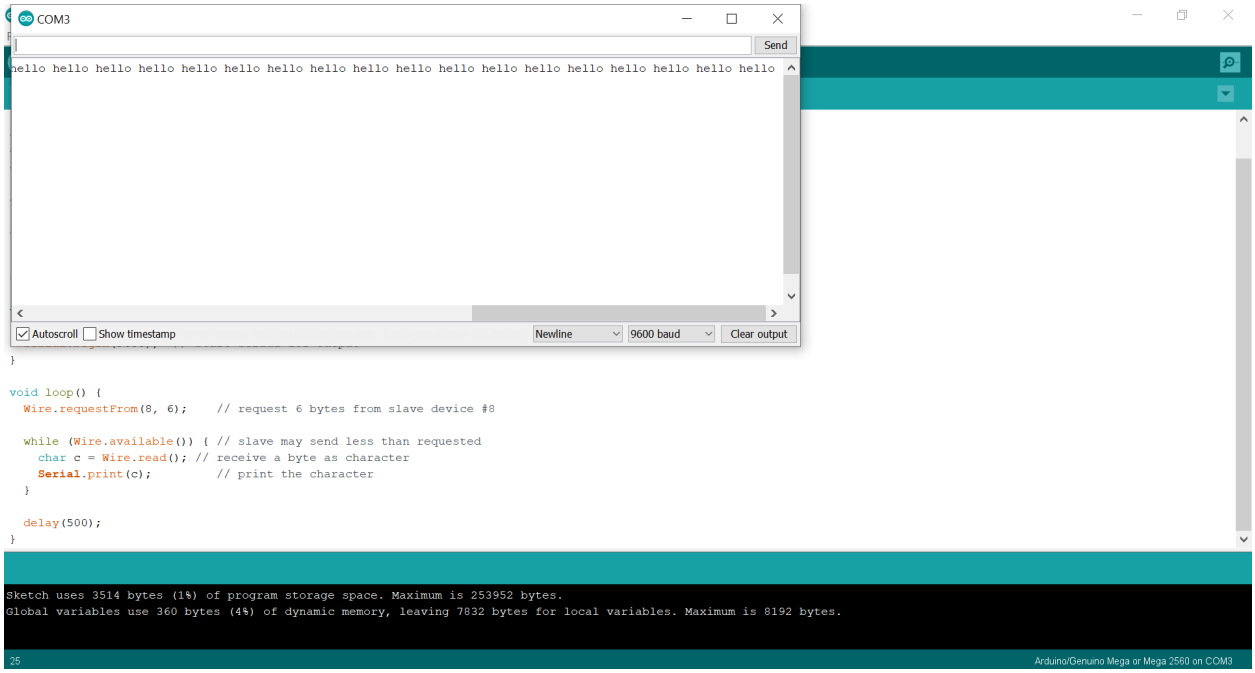


Figure 17: Serial Monitor

The third major test was the integration test which tested each aspect of the PID controller, the heating unit, set temperature, and LCD displaying all of the important parameters. This test is shown below through an image of the LCD screen which displays the set temperature "Set:" and the real temperature "Real temp:" The heating unit as seen in the picture never climbs to the desired temperature but did climb to 21 degrees Celsius. The display depicting this information can be found in Appendix A.

Test one performed extremely well and had the accuracy that was expected from research and communication with the client. The important requirement that was met here is the ability to read the correct real temperature through the k-type thermocouple.

Test two performed well throughout the entire test except for one aspect. The system did not have the ability to communicate between Arduinos and display it on the LCD screen

like the team had thought. The Arduino could transmit to the LCD screen or the other Arduino but not both. This was a small hindrance in our test that could have been changed by utilizing a Zigbee Arduino module for wireless communication. Unfortunately, this problem was not found in our research and the team paid for it toward the end of the project. The important requirement here was not met which was communication between Arduinos and displaying it to the LCD screen.

Test three underperformed during the testing phase. The PID temperature controller did not control the temperature to the accuracy that the team expected or wanted. This is mainly due to the fact that we had trouble controlling the power supply converter to apply the correct current and voltage to the SSR and heat the oven heater to the setpoint. The most important requirements were not met as this was the heater control test and we did not heat the oven to a temperature that was within 3 degrees of the setpoint.

b. Lessons Learned

In the testing of our project, we encountered a few roadblocks that were described in the Major Tests and Analysis of Results sections of this document. Due to issues we faced while testing, we had to change and get rid of some of our requirements to complete the project. Also, during testing we had to modify different aspects of our design. We initially were working with an I2C LCD screen but had to change to a regular LCD screen when we faced issues with the first one. After we began testing, we needed more space on our Arduino Board so had to change back to the I2C LCD screen to save space on the board for other parts of the project. Although we modified our design, we still did not have space on our board to operate the LCD screen and use Arduino to Arduino communication as they required the same pins. As this was done at the end of our testing, we did not find a quick and easy solution. In the future we could have experimented with different forms of Arduino communication that were not wired.

We did a number of regression tests with our design, as the project was dependent on the code working properly with our hardware. This is how we were able to fix bugs that came up as we were completing our other testing. For example, this was done when we encountered issues with our LCD screen when we were testing our Arduino to Arduino communication. The LCD screen worked before we set up the communication but would not as we were testing the functionality of the communication. As we were testing, the importance of testing procedures became apparent. There are aspects of the project that we would not have known to check on until working with the final product if we had not tested them, the main example being our communication system working with the LCD

screen. Testing was important to discover the issue and, with more time, find a different approach.

8. User Manual

a. Introduction

We are pleased that you have chosen Jumping Jacks for your business needs. There is a strong need for a heating and pressure system to conduct experiments in the Shuur Lab. We provide for you here a system designed to control the heating system and connect it to the pressure system of the reaction system. Some of the key highlights include: a display to show the desired and actual temperatures of the system, a temperature sensor to measure the temperature of the ovens, and Arduino communication to connect the new temperature system to the pressure system built last year. The purpose of this user manual is to help you, the client, successfully use and maintain the gas reaction system going forward. Our aim is to make sure that you are able to benefit from our product for many years to come.

The project focuses on making the Isotope Sample Preparation Lab more power efficient and up to date in terms of electronics. The project outline was to enhance, automate, and make a pressure and temperature monitoring system that the client uses to work on carbon samples to create needed isotopes. The client expressed issues of the old system which were: old parts being discontinued, old power supplies coming to an end, and the need for manual control throughout the process. The automation of the sampling process would help the client indefinitely as the reaction can take anywhere from 2-5 hours and has to be manually turned off using the current system. With the proposed solution this process would happen on its own after the pressure starts to level out for a prolonged period of time. The project would be able to read the pressure of 13 samples, 12 arranged by the client and a reference pressure. Although, having a pressure and temperature system work together is not uncommon; the niche aspect comes in with the magnitude the pressure and temperature systems have to be able to read and reach. The carbon samples will have to be heated up anywhere from 600 to 900 degrees Celsius. The need for this is due to carbon's ability to withstand extreme temperatures and turn into three usable isotopes for analysis.

We identified several important requirements that were the focus of our design and subsystems. These requirements included:

- The Arduino microcontroller will get inputs from the pressure and heating sensor that are attached to the vacuum chamber where the reaction occurs.
- The values from the pressure and temperature sensor are then displayed on an LCD screen for easy viewing.

- A user input will be needed for the “set temperature” which is also displayed on the LCD screen.
- The pressure sensor is used to monitor the reaction and determine whether the heating block should be on or off based on its readings.
- Store the stop time of the reaction and print it on the LCD screen.

The subsystems that we identified for this project are as follows: the input, the display, the PID controller, storing data, and the vacuum chamber.

The first subsystem is for the input. In this subsystem, the user will set the temperature between 550-950 degrees Celsius with push buttons. The set temperature value will then be sent to the PID Controller. The second subsystem is for the display. In this subsystem, the PID Controller will have the actual temperature, set temperature, and pressure values and the stop time. This information will be sent to the LCD screen to be displayed. The third subsystem is for the PID controller. In this subsystem, the user input as well as the information from the vacuum chamber will be sent to the Arduino Uno. The Arduino will then send information to the display, tell the heater if it needs to adjust. The fourth subsystem is for storing data. In this subsystem, the PID Controller code determines which information needs to be saved i.e temperature, pressure, and stop time. Some of the saved information will be sent to the display later on. The last subsystem is for the vacuum chamber. The vacuum chamber is already created in the lab with the reaction occuring inside. The pressure sensor is attached to the vacuum chamber while the temperature sensor is integrated into the k-Type thermocouple. Information is then sent to the PID Controller.

The solution that our group identified was a proportional (P), integral (I), derivative (D) (PID) temperature controller that works in conjunction with a pressure system. The project would be able to read the pressure of 13 samples, 12 arranged by the client and a reference pressure. The Arduino microcontroller will get inputs from the pressure and heating sensor that are attached to the vacuum chamber where the reaction occurs. The project then uses the PID temperature controller to reach a desired “setpoint” chosen by the user and left alone while the reactions go to completion. The values from the pressure and temperature sensor are then displayed on an LCD screen for easy viewing. A user input will be needed for the “set temperature,” which is also displayed on the LCD screen. The pressure sensor is used to monitor the reaction and determine whether the heating block should be on or off based on its readings. Lastly, the stop time of the reaction will be stored and printed on the LCD screen.

This design was comprehensive as it would be able to meet each of the requirements set both by the client as well as our team. In our testing, we tested each of the major components of the project, the LCD screen, measuring different temperatures, and setting the temperature of the oven. Although not all of the tests passed, we were able to cover each of the main components of the design, following the steps taken in planning to find

a solution. In this we also attempted to cover more than the set requirements we initially had, adding Arduino to Ardunio communication to connect to the previous team's project as well to create an entire, connected system.

b. Installation

First, it is important to know that the code we have used for the project can be found in Appendix B, Appendix C, and Appendix D. This code allows for the LCD screen to display the measured temperatures, for the PID controller to set the temperature of the system, and for communication between the Arduinos. The only user interface with the device will be the use of the two buttons to set the temperature of the oven, as it ideally turns off as needed when connected to the pressure aspect of the project. This is described more in the section below.

Installation of the device should be relatively simple, as there is an existing device in place that would just need to be replaced. The device would need to be connected to the power source supplied with the oven and existing parts given by the Shuur Lab. THis would need to be connected to the Arduino to keep it powered, and also connected to an outlet as it is now for the ovens. When moving the project, you would need to be delicate so as to not displace the wires or any of the various parts. The code is currently uploaded to the Arduino, but once all aspects of the code are working properly in the future, all code would need to be saved into a single file and uploaded using Arduino IDE to the Arduino itself. The Arduino is an Ardunio Mega so that should be selected when installing the software. The device would then be able to be connected directly to the current ovens and should work as needed. The below image shows the current setup of the system in the Shuur Lab.

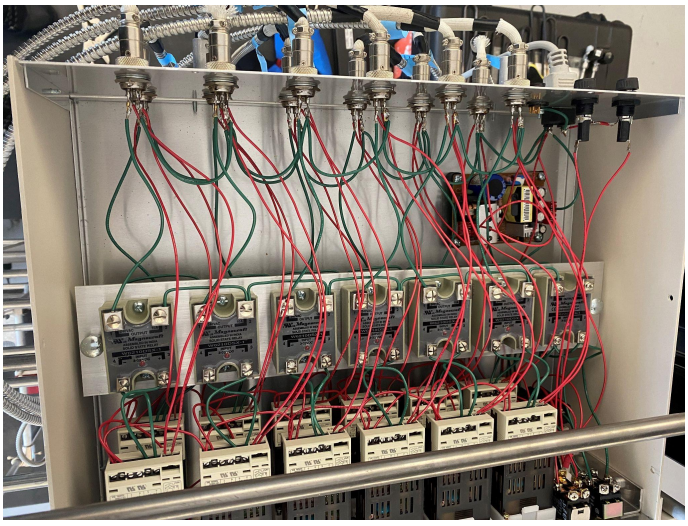


Figure 18: Current Setup

c. Configuration and Use

Once the device is properly installed, there is little user interaction that will be required. The only user interface is the use of the buttons that will control the temperature of the system. There are two buttons, one which will increase the temperature of the system, and a second button that will decrease the temperature of the system. Both operate in increments of 5 degrees Celsius. Once the proper temperature is selected using the buttons there are no longer any actions that need to be taken by the user. The temperature of the oven will operate automatically.

d. Maintenance

There are few parts or pieces of code in this system that would need to be maintained. The main component, the thermocouple, can be easily replaced if necessary. The current part would need to be taken out of the system, then replaced in the same spot, reattaching the wires the old part had used to the new part. This same procedure can be done for each of the components in the system; the buttons, resistors, or the solid state relay. As any of the parts are replaced with equivalent parts, there would be no need to change the code on any of the Arduino boards. Below are links to replacements for hardware parts.

For the solid state relay:
https://www.omega.com/en-us/control-monitoring/relays/solid-state-relays/p/SSRL240-660

For the thermocouple:
https://www.omega.com/en-us/control-monitoring/relays/solid-state-relays/p/SSRL240-660

For the LCD screen:
https://www.omega.com/en-us/control-monitoring/relays/solid-state-relays/p/SSRL240-660

For the buttons:
https://www.mouser.com/ProductDetail/SparkFun/COM-10302?qs=WyAARYrbSnaYi0oOM0cIVQ%3D%3D&gclid=CjwKCAjwlID8BRAFEiwAnUoK1cGx_BE_-M9dirYZ19XEcF-GA3egh7usD30ecMgm6v11OBhQlvqTCxoCz3UQAvD_BwE

e. Troubleshooting Operation

During testing there were a few different issues that arose and could occur again. The first is the LCD screen. In general LCD screens are reliable, but we had multiple issues with the two LCD screens that we had used. This would be a simple item to fix, as it would just require replacing the LCD screen in the current model with the same screen. This would require removing the current screen and placing the correct wires in the correct spots for the new screen. At this moment in time, there are wires connected to the ground and power portions of the breadboard and Arduino. These wires will not be changed and can be connected to the power and ground pins in the LCD screen, whose marks can be seen on the side of the LCD screen. The remaining pins are connected using wires to pins 12, 11, 6, 5, 4, and 3 on the Arduino respectively as pins RS, Enable, D4, D5, D6, and D7.

A second problem that could arise is the accidental removal or breaking of the buttons used by the user, as they would be used often and could break down. The fix for this would be the same as the LCD screen, with the omission of the wires needed. The replacement here would be to buy a new button (whose link is listed in the above section) and place it in the same spot as the current button.

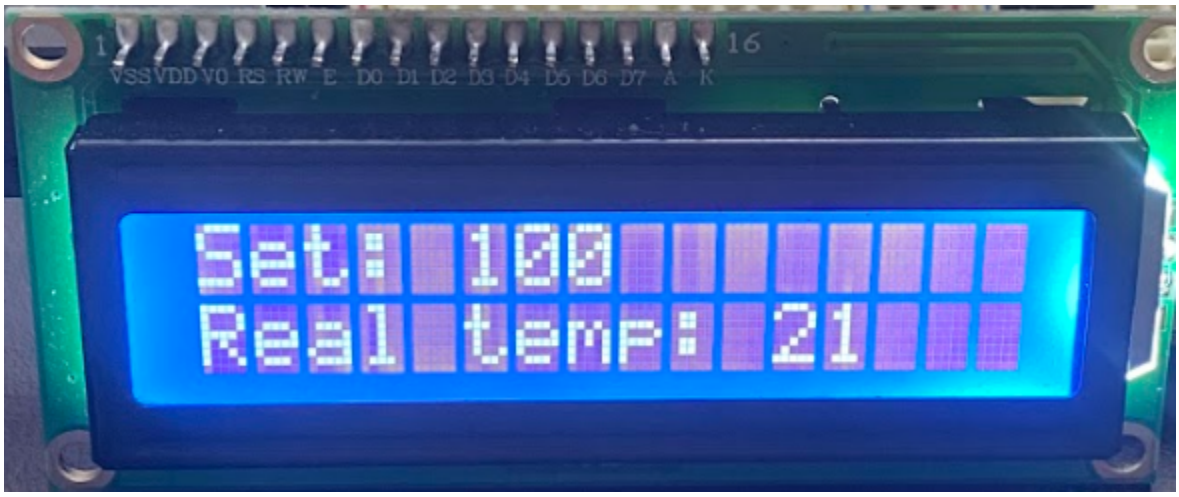
The last issue that we could foresee would be the moving of wires from the Solid State Relay, as we can across that problem when we moved the project too much in the testing phase. This would be solved by using a small screwdriver to loosen the screws on the solid state relay. You would then need to place the ends of the wires without the coating back into the space directly under the screw. While holding the wire there, use the screwdriver to tighten the screw again with the wire underneath.

f. Conclusion

We wish you years of productive use of this system that we have laid out. We also hope this manual has helped in the building and operation of this new system and will help in replacing the current parts that you have. And we are happy to have been able to help with your problem.
Below are our email addresses if you wish to contact us in the future, we would be happy to answer any questions with the system as they arise:
kah635@nau.edu, rdh258@nau.edu, yw249@nau.edu, eaz34@nau.edu

9. Appendices

Appendix A



This display shows the set and real temperatures for our system. Where the set temperature is given by the user, and the real temperature is measured using the k-type thermocouple.

Appendix B

The code below works to allow Arduino to Arduino communication. The setup of this communication had not fully been fleshed out as this was one of the issues that we came across in testing.

```
#include <Wire.h>
#include <LiquidCrystal.h>
int LED = 13;
int x = 0;
int Contrast = 75;
LiquidCrystal lcd(12, 11, 6, 5, 4, 3);

void setup() {
pinMode (LED, OUTPUT);
// Start the I2C Bus as Slave on address 9
Wire.begin(9);
// Attach a function to trigger when something is received.
Wire.onReceive(receiveEvent);
analogWrite(7,Contrast);
lcd.begin(16, 2);

}
void receiveEvent(int bytes) {
x = Wire.read(); // read one character from the I2C
}
void loop() {
//If value received is 0 blink LED for 200 ms
if (x == '0') {
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Here");

digitalWrite(LED, HIGH);
delay(200);
digitalWrite(LED, LOW);
delay(200);
}
//If value received is 3 blink LED for 400 ms
if (x == '3') {
lcd.clear();
```

```
lcd.setCursor(0,0);
lcd.print("Hear");
digitalWrite(LED, HIGH);
delay(400);
digitalWrite(LED, LOW);
delay(400);
}
}
```

MASTER

```
// Include the standard Wire library for I2C
#include <Wire.h>
int x = 3;
void setup() {
// Start the I2C Bus as Master
Wire.begin();
}
void loop() {
Wire.beginTransmission(9); // transmit to device #9
Wire.write(x); // sends x
Wire.endTransmission(); // stop transmitting
//x++; // Increment x
//if (x > 5) x = 0; // reset x once it gets 6
delay(500);
}
```

Appendix C

The below code is for the temperature sensor and displaying the found temperatures on the LCD screen. This is working properly in our system.

```
/*    Max6675 Module  ==>   Arduino
 *    CS             ==>     D10    \\cs = chip select??
 *    SO             ==>     D9    \\so = standard output
 *    SCK             ==>     D13   \\sck = serial clock
 *    Vcc            ==>     Vcc (5v)
 *    Gnd             ==>     Gnd      */

//LCD config
#include "max6675.h"
#include <Wire.h>
#include <LiquidCrystal.h>
int Contrast=75;
 LiquidCrystal lcd(12, 11, 6, 5, 4, 3);



int thermoDO = 9;
int thermoCS = 10;
int thermoCLK = 13;

//Start a MAX6675 communication with the selected pins
MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);
int maximum_firing_delay = 7400;


unsigned long previousMillis = 0;
unsigned long currentMillis = 0;

int temp_read_Delay = 500;
int real_temperature = 0;
int setpoint = 100;
bool pressed_1 = false;
bool pressed_2 = false;

 void setup()
 {
```

```
    analogWrite(7,Contrast);
     lcd.begin(16, 2);
  }

void loop() {
    currentMillis = millis();

// Max 6675 can only read every 500ms or ½ sec
  if(currentMillis - previousMillis >= temp_read_Delay){
    previousMillis += temp_read_Delay;              //Increase the previous time for next loop
    real_temperature = thermocouple.readCelsius();  //get the real temperature in Celsius
degrees


    //Print the values on the LCD
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Set: ");
    lcd.setCursor(5,0);
    lcd.print(setpoint);
    lcd.setCursor(0,1);
    lcd.print("Real temp: ");
    lcd.setCursor(11,1);
    lcd.print(real_temperature);

  }
}
```

Appendix D

The following code is for the PID control. This was another aspect of the project that we were unable to make work properly. The hardware setup works, but there are still some issues with the code that need to be worked out.

```
/********************************************************
* PID RelayOutput Example
* Same as basic example, except that this time, the output
* is going to a digital pin which (we presume) is controlling
* a relay.  the pid is designed to Output an analog value,
* but the relay can only be On/Off.
*
*   to connect them together we use "time proportioning
* control"  it's essentially a really slow version of PWM.
* first we decide on a window size (5000mS say.) we then
* set the pid to adjust its output between 0 and that window
* size.  lastly, we add some logic that translates the PID
* output into "Relay On Time" with the remainder of the
* window being "Relay Off Time"
********************************************************/
#include <max6675.h>
#include <PID_v1.h>
#define RelayPin 8

//Define Variables we'll be connecting to
double Setpoint, Input, Output;

//Specify the links and initial tuning parameters
PID myPID(&Input, &Output, &Setpoint,9.1,0.3,1.8, DIRECT);

int WindowSize = 1000;
unsigned long windowStartTime;

//31855 stuff
int thermoCLK = 9;
int thermoCS = 22;
int thermoDO = 13;
//int backLight = 13;    // pin 13 will control the backlight
MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);
```

```
void setup()
{

  Serial.begin(9600);
  windowStartTime = millis();

  //initialize the variables we're linked to
  Setpoint = 100;


  //tell the PID to range between 0 and the full window size
  myPID.SetOutputLimits(0, WindowSize);

  //turn the PID on
  myPID.SetMode(AUTOMATIC);

  pinMode(RelayPin, OUTPUT);
}

void loop()
{
  //pinMode(RelayPin, OUTPUT);
  Input = thermocouple.readCelsius();

  myPID.Compute();
  Serial.print("INPUT:");
  Serial.println(Input);
  Serial.print("OUTPUT:");
  Serial.println(Output);
delay(2000);
//have delay at 500 when running
  /************************************************
   * turn the output pin on/off based on pid output
   ************************************************/
  if(millis() - windowStartTime>WindowSize)
  { //time to shift the Relay Window
    windowStartTime += WindowSize;
  }
  if(Output > millis() - windowStartTime) digitalWrite(RelayPin,HIGH);

  else digitalWrite(RelayPin,LOW);
}
```