Automated Frozen Pipe Burst Prevention System

ECE 445 FINAL REPORT – SPRING 2023

Project #22

Neha Vagvala, Benedicta Udeogu, Ethan Zhang

Professor: Arne Fliflet

TA: Prannoy Kathiresan

Abstract

The purpose of this report is to introduce and summarize the design and findings of our ECE 445 Senior Design Project: Automated Frozen Pipe Burst Prevention System. The result of our project is a fully functional prevention system that automatically triggers two preventative methods upon detecting the water temperature at or below the threshold of 55 degrees Fahrenheit: the valve subsystem and the notification subsystem. The valve subsystem sends a trickle of water through the pipe at programmed intervals to help prevent heat loss from exceeding flow rate, and the notification subsystem sends a push notification to the designated resident's smartphone to alert them that the given pipe is at risk. The following were the three high-level requirements that we successfully demonstrated:

1. The system is triggered only when the temperature sensor output reads at or below 13.0 °C (55 °F).
2. A push notification is sent to the mobile device using Wi-Fi upon the Raspberry Pi receiving a low temperature reading.
3. Water is deposited through the pipe over the course of 5 seconds, so that the movement of trickling water prevents the pipe from freezing.

Contents

1. Introduction

The purpose of this project is to come up with an automated solution to prevent frozen pipe bursts. It is estimated that an average of over 250,000 homes will suffer damage from frozen and burst pipes each year. The damage is estimated to rack up to $400–500 million each year. To further highlight the fragility of frozen pipes, even a rupture as small as 1/8th of an inch can release up to 250 gallons of water per day.

Frozen pipe bursts occur in cases of plummeting temperatures outdoors (or due to the thawing of ice thereafter). Water expands as it transforms into its solid state (ice), increasing the pressure inside the pipe until it ruptures.These unfortunate bursts can be attributed to a myriad of factors, such as insufficient insulation or lack thereof, the household resident forgetting to set the household thermostat to 55 °F, and/or the household resident forgetting to open the tap to allow a small trickle through.

That is why we have devised an automated two-part solution that offers preventative measures, both triggered by the ATmega328P microcontroller upon crossing below the threshold temperature of 55 °F. The first is a valve that releases a trickle of water into the pipe so that the movement prevents the water from solidifying, and the second is a notification that is sent to the designated resident’s phone indicating the current pipe temperature detected and alerting them that their pipe may be at risk of bursting if outdoor weather conditions continue to worsen and it is left unattended. This provides the resident with the peace of mind of knowing exactly when their pipes may be at risk and enables them to take action if necessary before it's too late, in case they left the house without setting the household thermostat to 55 degrees or leaving the tap open so that a trickle of water can flow through. Our system is also natural resource and budget conscious, as it is only triggered when needed, as opposed to the resident leaving the tap open for extended periods of time, which results in unnecessary water wastage and excess utility costs.

2. Design

2.1 Block Diagram & Design Procedure

Pictured below is the block diagram for our project. It consists of three subsystems: 1) the microcontroller subsystem, 2) the valve control subsystem, and 3) the notification subsystem. These three subsystems integrate together cohesively by transmitting data signals and power between each other, as denoted by the colored arrows. In particular, the microcontroller interfaces with the temperature sensor via the 1-Wire protocol, the LCD display to display the temperature reading, and the transistor in order to activate the electronic solenoid valve, which uses 12V DC power. The Raspberry Pi of the notification subsystem also interfaces with the temperature sensor via its GPIO pins and the 1-Wire protocol, and connects wirelessly to the internet in order to send the push notifications. Each individual subsystem is discussed in further detail in subsequent sections.

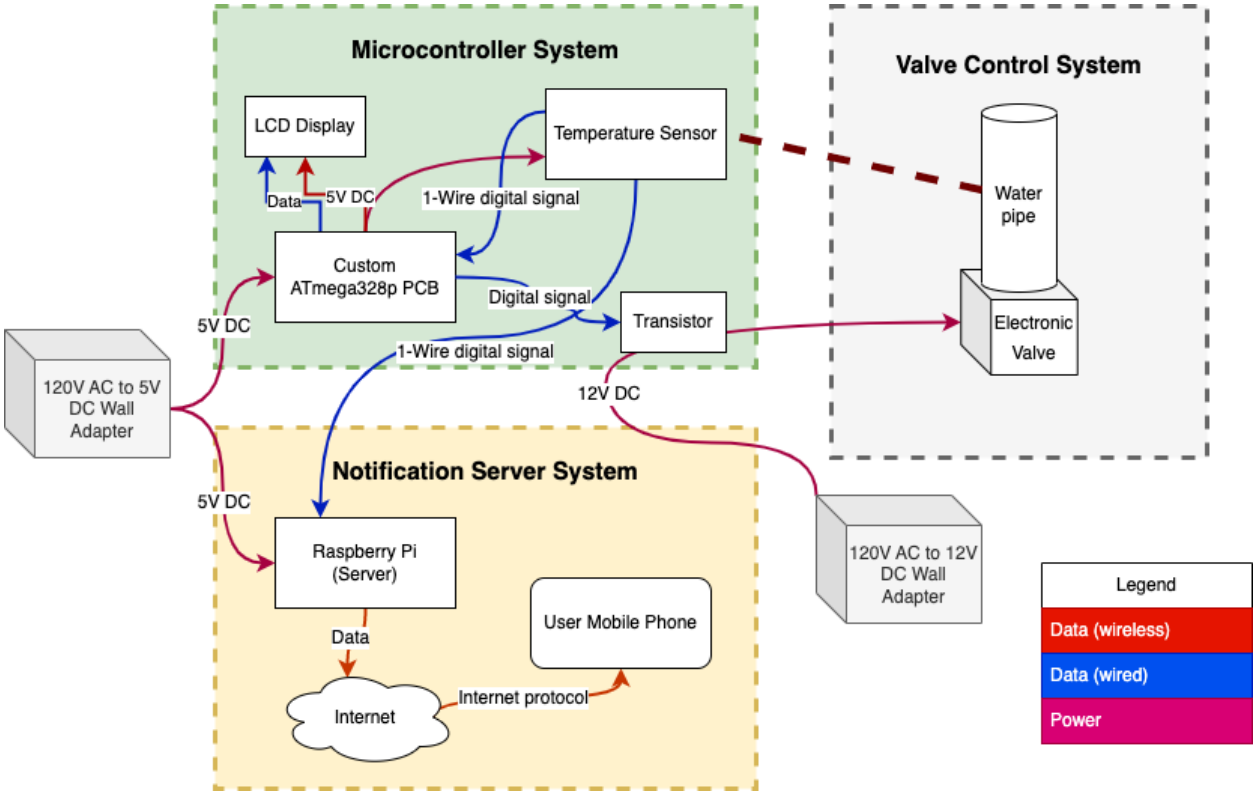

Figure 1. Block diagram

In regards to the design procedure, multiple options were explored for the implementations of each subsystem prior to a final decision being chosen. For the microcontroller subsystem, we considered multiple different options for several components, most notably the temperature sensor. We had initially considered a temperature sensor which was not waterproof and would thus be more appropriate to be affixed to the exterior of the pipe in order to measure the change in temperature, however, after consulting with the ECE Machine Shop staff, we determined that for this to be viable it would require a more thermally conductive pipe material such as copper, which would not only make the construction more complex but would also be less faithful to real life scenarios, as the majority of plumbing in modern homes utilizes PVC pipes. As a result, in our final design we opted for the DS18B20 Temperature Sensor, which is a fully waterproof temperature probe which we could insert directly into the section of pipe for which we desire the water temperature to be read.

Accordingly, for the valve control subsystem, we opted for our apparatus to utilize PVC pipes as opposed to copper or galvanized steel pipes, as we had been considering in the early stages of planning our project. As a result, we ordered a solenoid valve that is compatible with PVC pipe, which was also more affordable than alternatives.

Lastly, for the notification subsystem, we had settled upon using a Raspberry Pi in our initial design due its in-built wi-fi functionality as well as easy programmability, but we also considered using a smaller and less powerful and expensive device with wi-fi capabilities, such as an ESP32. However, based on online research and the experiences of peers, we determined that using a more niche and less flexible devices such as these would yield a higher probability that our notification subsystem would not work at all for our demonstration, especially given the requirement of connecting to the university enterprise wi-fi, which requires additional authentication measures. As a result, we decided to continue using the Raspberry Pi for our project, as it would result in the highest likelihood of success.

2.2 Design Details

2.2.1 PCB Schematic

The project ran on a custom-made PCB that contained an ATmega328p microcontroller.

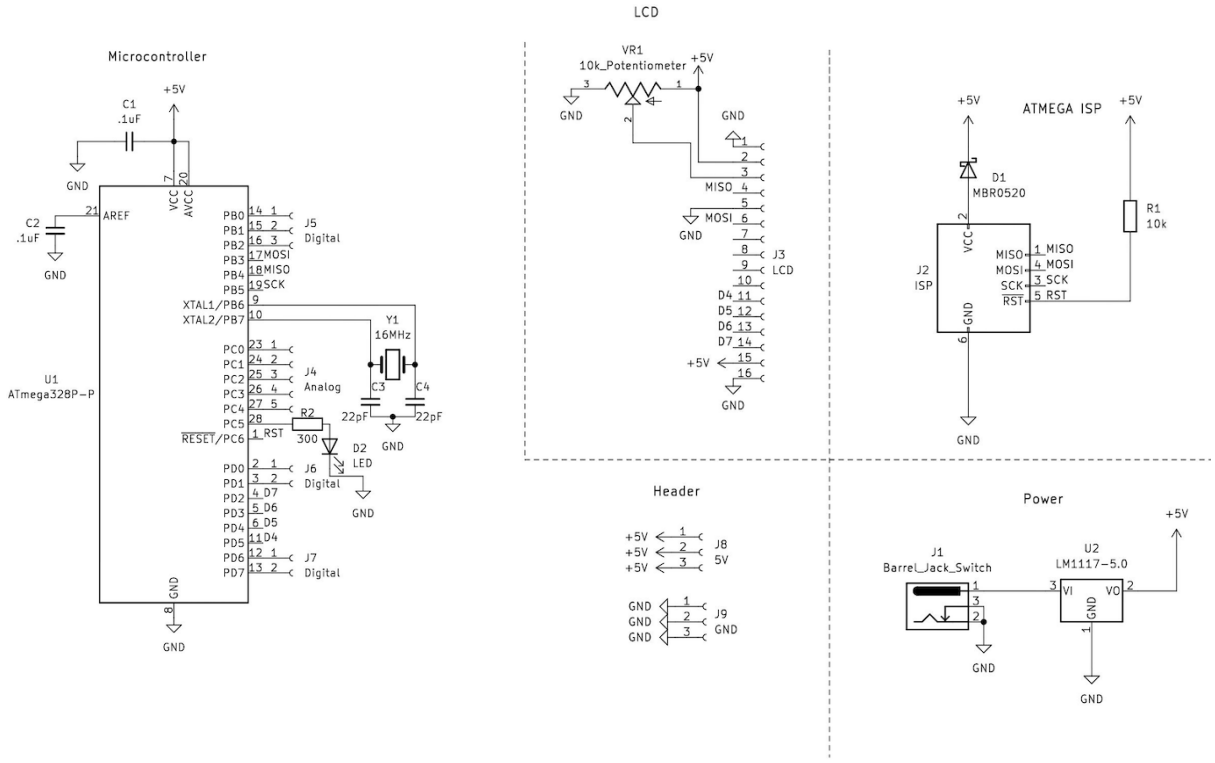

Figure 2. PCB Schematic

The schematic of the microcontroller features an external 16 MHz clock oscillator in charge of the data control speed. In comparison to the internal 8 MHz clock, the 16 MHz oscillator provided data transfer speeds twice as fast, which in turn increased power savings. As the circuit was running at 5V rather than the alternating 3.3V, the 16 MHz clock was necessary due to a higher current draw. Other components in the schematic are the LCD, operating in 4-bit data mode; the 6-pin ISP programmer used to flash the microcontroller; and the power supply, a 5V wall adapter fed through a 5V linear voltage regulator.

Figure 2 shows a 3D rendering of the PCB. A preference for through-hole (THT) components was favored over surface-mount (SMD) parts as THT components were easier to solder and came in bigger sizes while still providing the same functionality. Lastly, the custom PCB was

created with the Arduino Uno as a reference because the Arduino Uno was the original fallback plan should the custom PCB not program or operate correctly.

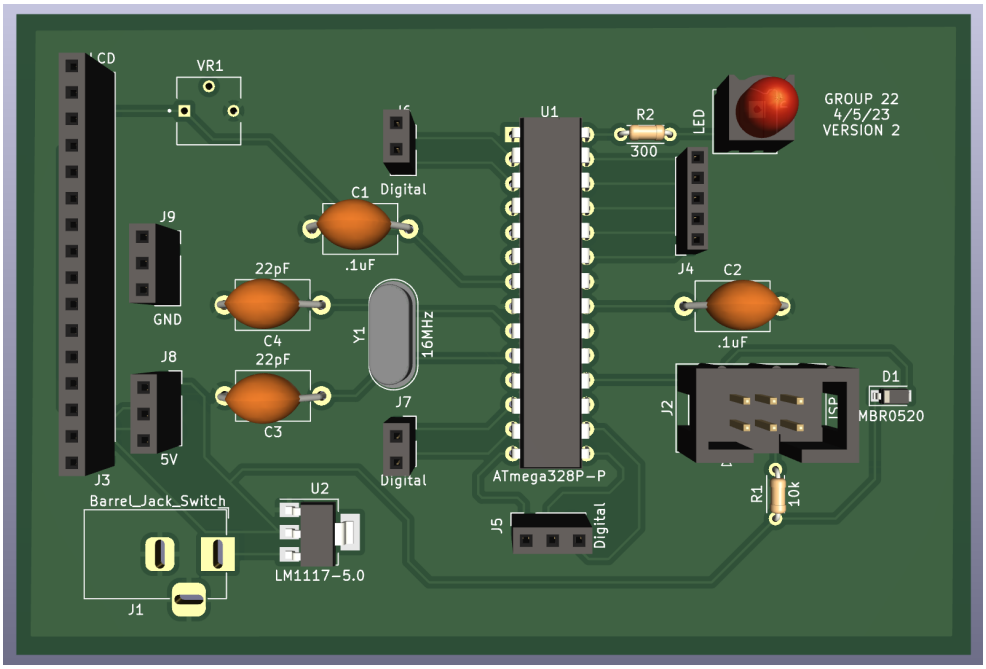

Figure 3. 3D Rendering of PCB

2.2.2 Microcontroller Subsystem

The implementation of this subsystem is primarily composed of two critical components: the ATmega328P microcontroller and the waterproof DS18B20 [1] digital temperature sensor. The ATmega328P MCU serves as the heart of the project and is utilized to interface with the DS18B20 temperature sensor, the LCD display, the plastic water solenoid valve, and the Raspberry Pi 4. The temperature sensor utilizes the 1-Wire protocol to be able to communicate with the microcontroller and is powered via a 5 VDC power adapter. More specifically, the temperature sensor produces a 9 to 12 bit output that is then converted into degrees Celsius and degrees Fahrenheit using the OneWire [2], DallasTemperature [3], and LiquidCrystal [4] libraries. The temperature sensor takes 750 µs to query the water temperature within the pipe. Once the MCU receives an output from the temperature sensor that the water temperature has

crossed below the threshold of 55 °F, the valve control and notification subsystems will be triggered.

```
sensors.requestTemperatures(); // Send the command to get temperatures
tempC = sensors.getTempCByIndex(0); // get the temperature - from the first sensor by default
tempF = sensors.toFahrenheit(tempC);
delay(2000);
```

Figure 4. Code to Process Waterproof DS18B20 Digital Temperature Sensor Output

2.2.3 Valve Control Subsystem

The implementation of this subsystem consisted of the use of a normally closed, 12 V, ½" in diameter, solenoid valve to pass water through a ½" in diameter PVC pipe. The signal to open the valve was provided by the 5V microcontroller, as that was the unit reading the temperature sensor outputs. To enable the valve using the ATmega, a NMOS transistor was used. The gate pin of the transistor was connected to a digital I/O pin of the ATmega328p, and when the MCU received a sufficiently low temperature reading, a HIGH signal was sent to the gate pin. The transistor, acting like a switch, would complete the solenoid valve circuit and allow the 12 V to pass through so that water could be deposited into the PVC pipe.

```
digitalWrite(solenoid, HIGH);
cnt = 0;
while(cnt < 5){
  lcd.setCursor(1, 1);
  lcd.print("VALVE OPEN  ");
  lcd.setCursor(13, 1);
  lcd.print("0");
  lcd.setCursor(14, 1);
  lcd.print(5 - cnt);
  delay(1000);
  cnt++;
}
digitalWrite(solenoid, LOW);
```

Figure 5. Code to Open Valve for 5 s

To control how long the valve stayed open, a 5 s delay was applied, as shown in the while loop in figure 5, before the gate pin was set to LOW. The system then returned to its original state of measuring the temperature within our water reservoir, waiting to repeat the valve process once the temperature dropped below 55 °F.

Additional software used for the solenoid valve was the inclusion of the OneWire and DallasTemperature libraries to aid in the data conversion of the 1-Wire temperature sensor. The LiquidCrystal library was also utilized, but to interact with the LCD display.

The hardware for this project consisted mostly of I/O communication between the microcontroller, temperature sensor, solenoid valve, and LCD display; the MCU would read the temperature and display it on the LCD. If the temperature was low enough, the valve would be engaged, and messages depicting the action of the valve would show up on the LCD display. This process would repeat for as long as the system was powered.

2.2.4 Notification Subsystem

The notification subsystem runs on a Raspberry Pi 4 single-board computer, running the standard Raspberry Pi Linux OS. Like the ATmega328p PCB, the Raspberry Pi accepts 5V DC as its power input. The DS18B20 temperature probe is connected to the Raspberry Pi via its GPIO pins, and like the microcontroller utilizes the 1-Wire protocol for digital data communication. The Raspberry Pi is connected to the user's home wi-fi network in order to send push notifications over the internet, and it can also easily be configured to connect to enterprise networks (such as IllinoisNet).

Both the temperature reading and notification sending are handled by a Python 3 script, which runs continuously on the system and is automatically launched upon boot using a system daemon. We are using the Telegram API to deliver the push notifications to the user, as it is reliable, fast, and the client is available on all mobile and desktop platforms. The bot API token and ID are stored securely in a separate YAML file on the system, and are needed to construct the URL which is then used to send the message containing the temperature to the user [5]. The message is only sent once the temperature reading from the sensor drops below the set threshold, which in the case of our demonstration is 55°F in order to reliably prevent water from freezing in the pipe, though this value can be adjusted as needed according to the user's preferences and needs. The notification is thus sent in sync with the valve opening activation. A flowchart diagram depicting the control flow of the script is shown below.

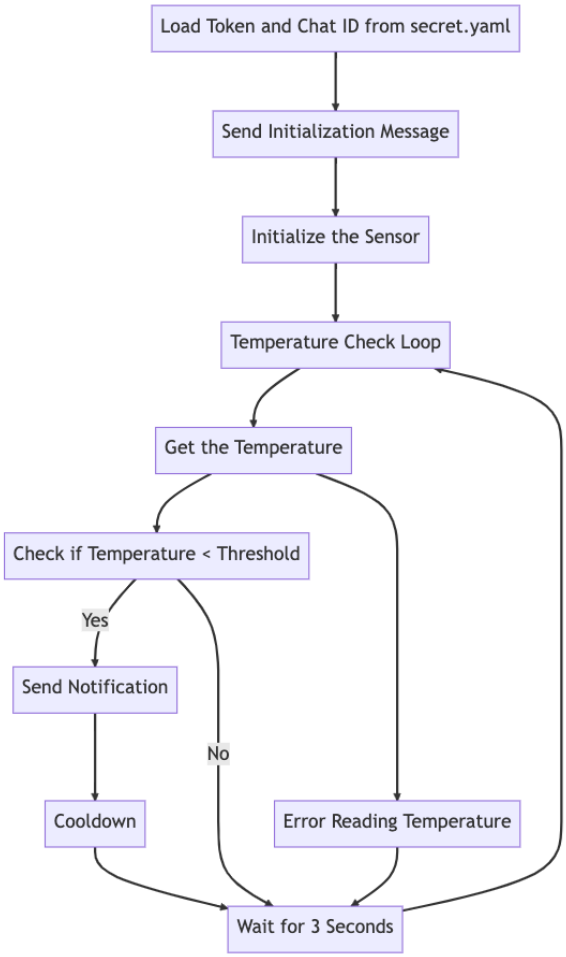

Figure 6. Diagram of the control flow of the notification script

The Python script utilizes the *requests* and *w1thermsensor* libraries for sending the push notifications and reading the temperature sensor, respectively. The push notification is sent via the Telegram API to the user's mobile phone, providing them with a real-time update on the status of their pipe burst prevention system, regardless of where they are in the world.

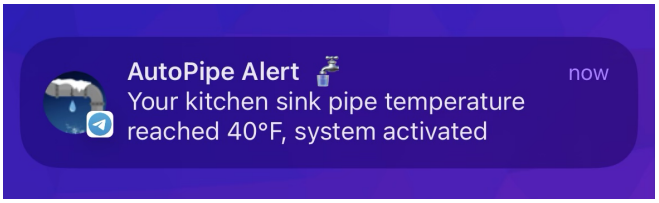

Figure 7. Example iOS push notification from a test run (with threshold set to 40°F)

3. Design Verification

Testing of the completed project involved finding the optimal refresh rate of the DS18B20 digital thermometer probe in order to prevent read interference on the one-wire data bus. Interference was likely to occur if the ATmega328p microcontroller attempted to read from the bus at the same time the Raspberry Pi attempted to read. Solutions included increasing the read latency and decreasing the refresh rate of the sensor. To increase the read latency, the delay between each request was increased from 1 s to 2 s.

As mentioned earlier, the DS18B20 temperature sensor can convert data anywhere from 9 to 12 bits in value. Our team considered reducing the refresh rate of the temperature sensor by reading at 9 bits as opposed to 12 bits to make the sensor 2^3 times faster. The result would theoretically be less wait time between each request of the sensor, but it was irrelevant as we had added a manual delay to the software. The preferred method was an increased latency of 2 s to give the microcontroller and microprocessor enough time to both read the current temperature. With the increase in wait time, we observed that the function meant to request the temperature was no longer returning a device error message but instead would always return the current temperature. Instead, we saw the ATmega328p and Raspberry Pi working in parallel.

Provided in the design review was Table 1, the Requirements and Verification Table. The requirements were meant to assure that the major components would work as they should. The major components included the microcontroller, temperature sensor, and solenoid valve. By the completion of our project, we had found success with all components as well as the requirements and verifications for each.

Table 1 – Requirements and Verifications

Component	Requirements	Verification
NMOS Transistor & Solenoid Valve	1. Provide 12 VDC +/- 10% from a 10.8 VDC to 13.2 VDC solenoid valve regulated by a 5 VDC microcontroller.	1. Using a NMOS transistor, connect a digital I/O pin from the ATmega328p, set as an output, to the "Gate" pin of the transistor and only close (set pin to HIGH) the gate when the temperature reading reaches threshold. Observe the valve open during this time.
DS18B20 Digital Temperature Sensor Probe	1. The system should only be triggered between a temperature of 13.0 °C and 14.0 °C, accounting for the +/- 0.5 °C tolerance of the DS18B20 Digital Thermometer. 2. Temp sensor conversion takes 750 s. Atmega328p and Raspberry Pi reading the DS18B20 temperature sensor do not cause read failures.	1. Measure the temperature of the water in the reservoir using an alternate thermometer and compare the result to the output of the DS18B20 digital thermometer on the LCD. 2. Add a delay (2000**)** after a read of the temperature sensor to see the removal of read failures as the ATmega328p and the Raspberry Pi attempt to read temperature versus when the delay is not there.
ATmega328p Microcontroller	1. The MCU burn bootloader works and is able to upload any sketch once flashed. Ability to interact with incoming data signals and sensors.	1. Connect an LED to an analog pin of the microcontroller. Set the LED to blink every second. After proper programming of the board through USBasp, one should see the LED blink in 1 s intervals.

4. Design Costs

4.1 Itemized Bill Cost

With our given budget of $150, the materials cost of this project came out to be **$90.18,** as specified in the itemized list in Table 2. Much of this cost was due to the need for a Raspberry Pi for Wi-Fi connectivity. In the conclusion chapter, a cheaper alternative, the ESP32 Wi-Fi module, is brought up as an alternative. If this chip were used in place of the microprocessor in our build, the itemized cost would drop to **$38.33**, making the project even more affordable.

Table 2 – Itemized List of Components and Costs

Description	Manufacturer	Quantity	Price	Link
IC MCU 8BIT 32KB FLASH 28DIP	Microchip Technology	1	$3.03	Link
WATERPROOF DS18B20 DIGITAL TEMPE	Adafruit Industries LLC	1	$9.95	Link
PLASTIC WATER SOLENOID VALVE - 1	Adafruit Industries LLC	1	$6.95	Link
LCD MOD 32DIG 16X2 TRANS YLW/GRN	Matrix Orbital	1	$9.53	Link
RASPBERRY PI 4 MODEL B 4GB SDRAM	Raspberry Pi	1	$55	Link
MOSFET N-CH 60V 30A TO220AB	STMicroelectronics	1	$1.54	Link
CRYSTAL 16.0000MHZ 20PF TH	IQD Frequency Products	1	$0.48	Link
LED 5MM VERTICAL GREEN PC MNT	Dialight	2	$1.81	Link
IC REG LINEAR 5V 800MA SOT223-4	Texas Instruments	1	$1.41	Link
RES 10K OHM 5% 1/4W AXIAL	Stackpole Electronics Inc	3	$0.10	Link
RES 100 OHM 5% 1/2W AXIAL	YAGEO	2	$0.17	Link
CAP CER 22PF 50V C0G 0402	TDK Corporation	2	$0.10	Link
CAP CER 0.1UF 4V X7T	TDK Corporation	5	$0.11	Link
TOTAL			**$90.18**	

4.2 Labor Costs

Using an ideal hourly rate of $45/hr, the cost of labor for our three person team, considering a completion time frame of two months, would come out to $20,250.

EQ 1. $\$45/hr\ x\ 2.5 hrs/day\ x\ 60\ days\ to\ complete\ x\ 3\ people\ =\ \$20,250$

There is also the consideration of the time the machine shop put into building our frame. After briefings of our ideal structure with one individual of the machine shop, they were able to complete our setup in two days. Using the same parameters used for the team's labor costs, replacing the days to complete with two and the number of people with one, the machine shop cost equates to $225.

EQ 2. $\$45/hr\ x\ 2.5 hrs/day\ x\ 2\ days\ for\ completion\ x\ 1\ person\ =\ \$225$

4.3 Total Project Cost

The total cost for this project, including an additional 10% in tax towards the bill of components, is **$20,574.19 total.**

EQ 3. $1.1\ *\ 90.18\ +\ 20,250\ +\ 225\ =\ \$20574.19$

5. Conclusions

5.1 Summary

Overall, all high-level requirements were executed successfully. The scope of the project remained the same throughout, and a system matching the proposal and the design document was delivered. To elaborate, our project repeatedly measured the water temperature, and upon a sufficiently low reading, a push notification was sent to the user's mobile device over Wi-Fi while the solenoid valve opened for 5 seconds and allowed a trickle of water through the ½” diameter PVC pipe.

5.2 Future Improvements

Potentially, some future improvements could be made to increase the elegance of the system. Suggested improvements are to replace the Raspberry Pi with an ESP32 microcontroller. The ESP32 module retails for $3.15, while the Raspberry Pi costs $55. Given the scope of this project, the Wi-Fi capabilities of the ESP32 MCU would handle the push notification script as well as the Raspberry Pi could. Had we been aware of this component prior to the end of our research and early build stages, we would have utilized it instead.

Another improvement that could be made is to shift from controlling the movement of the water in the users home pipes to focusing on raising the environment's temperature to prevent the need for water flow. This more hands-off and low-stakes option, while keeping with the original budget if the ESP32 module is used in place of the Raspberry Pi microprocessor, would include obtaining a smart thermostat for the home and creating an automation that sets the home's temperature to 55 °F if it were to drop below this threshold. The other portions of the project would remain the same, but now we eliminate the need for access to the water pipes beyond measuring the temperature of the pipe. The user would still receive a push notification if the temperature reading came back low enough to warrant a notification.

5.3 Ethical Considerations

In regards to ethics, our biggest concern was with the waterproofing of our project, but we took steps to keep with I.1 of the IEEE Code of Ethics [6] by utilizing water-resistant components when possible. Our project inherently attempts to create a sustainable alternative to current frozen pipe prevention measures by minimizing the use of resources such as water and heat. One aspect that could be considered unsafe was the proximity of our electrical components, including the PCB and Raspberry Pi, to the water reservoir and pipe for accurate measurements. Specifically, we were able to mitigate these risks by using a waterproof temperature probe that was 3 feet long and moving the LCD mount to the top of our project frame. See Figure 1 of the appendix. We also taped the PCB to the underside of the top portion of the frame, both for easy reach and to keep it away from the water escaping the valve. Further steps taken to prevent water exposure to the components were to move them to the side of our frame. Including proper enclosures for components that are not waterproof would provide an extra layer of protection. Additional safety precautions and procedures followed during the construction of this project included exercising caution when handling water around electrical wall outlets and other high-voltage equipment.

References

[1] “DS18B20 high temp waterproof digital sensors,” Mouser, https://www.mouser.com/new/sparkfun/ds18b20-waterproof-sensors/

[2]“1-Wire Protocol | Arduino Documentation,” *1-Wire Protocol | Arduino Documentation*. https://docs.arduino.cc/learn/communication/one-wire

[3] A. Industries, “Waterproof 1-Wire DS18B20 Digital temperature sensor,” *Waterproof 1-Wire DS18B20 Digital temperature sensor : ID 381 : $9.95 : Adafruit Industries, Unique & fun DIY electronics and kits*. https://www.adafruit.com/product/381

[4] “Liquid Crystal Displays (LCD) with Arduino | Arduino Documentation,” *Liquid Crystal Displays (LCD) with Arduino | Arduino Documentation*. https://docs.arduino.cc/learn/electronics/lcd-displays

[5] L. Z. Lin. Using python to send telegram messages in 3 simple steps. Medium. 03-Jun-2022. Retrieved March 27, 2023, from https://medium.com/codex/using-python-to-send-telegram-messages-in-3-simple-steps-419a8b5e5e2

[6] “IEEE Code of Ethics,” *IEEE - IEEE Code of Ethics*. https://www.ieee.org/about/corporate/governance/p7-8.html

Appendix

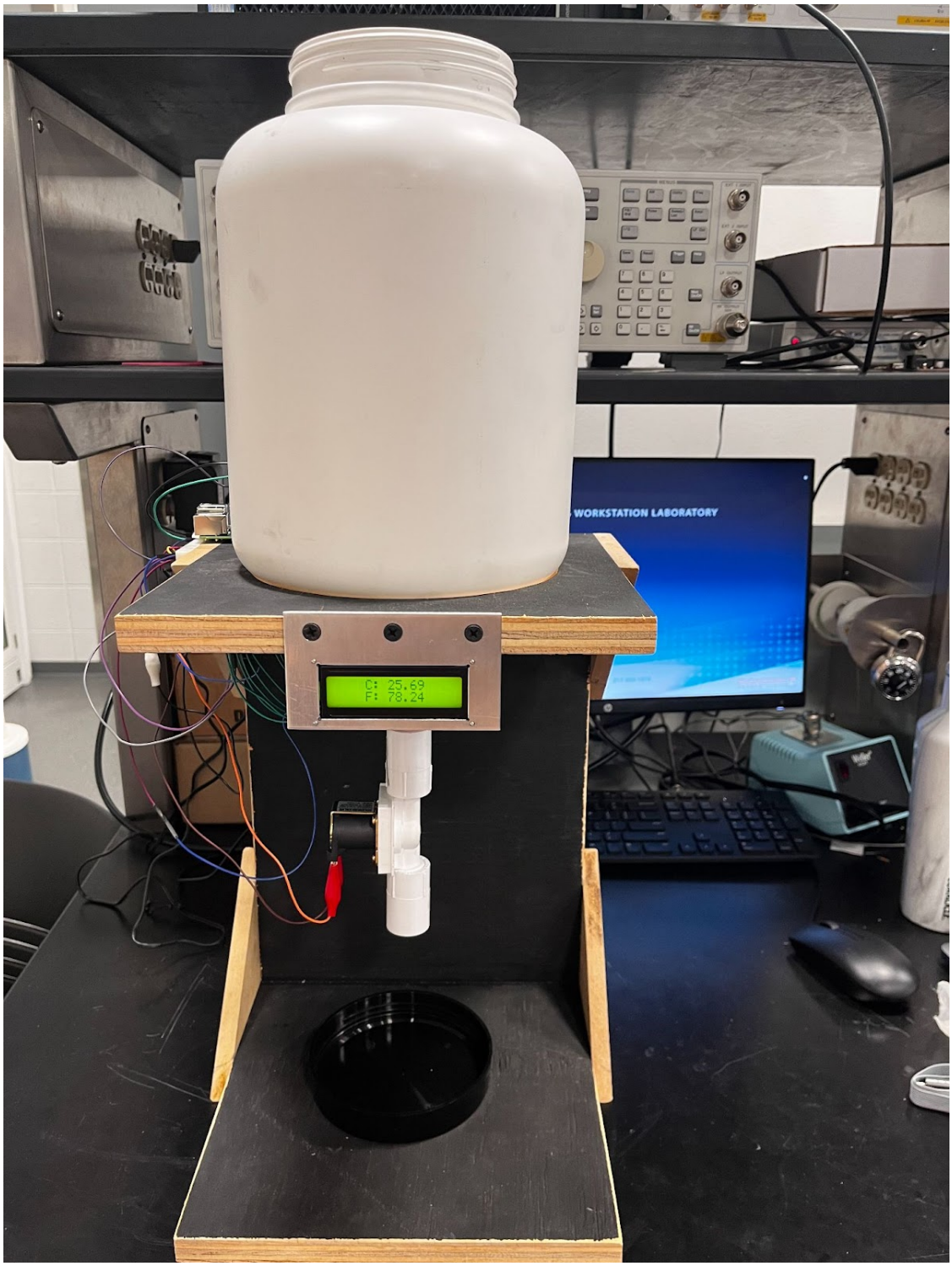

Figure 8. Front view of Project Setup

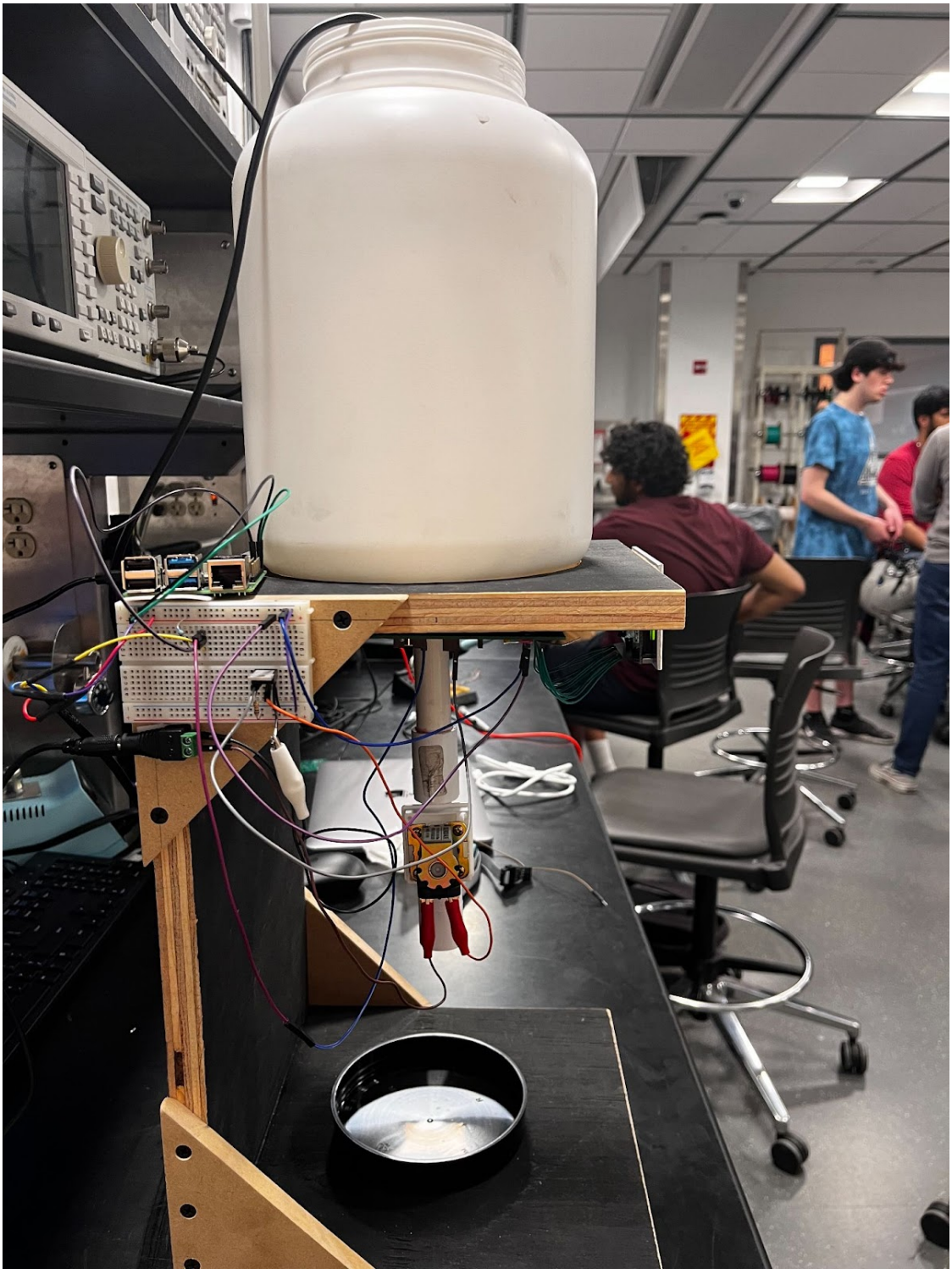

Figure 9. Side view of Project Setup