Smart Water Pipes

EECS 473
Fall 2018

Team members: Devesha Tewari, Rahul Gangwani, Chunan Ye, Yutian Chen, Carl Wu

I. Introduction and overview of project

As outdoor temperatures plummet, household pipes can be exposed to extremely cold conditions. This is experienced commonly by persons who are away from their home for extended periods during winter time such as in unattended vacation homes. Lower temperatures can cause the water in the pipes to freeze. This results in an increase in the pressure in the pipes, which can cause them to burst. The Michigan Committee for Severe Weather Awareness estimates that a quarter-million homes in Michigan are damaged each winter due to frozen water pipes, while an eighth-inch crack in a pipe can cause up to 250 gallons of water to leak a day[1]. This can lead to severe property damage. According to ACE Home Services, such a water leak can cost anywhere from \$5,000- \$50,000, plus plumbing costs to repair the pipes[2].

One common workaround to this problem is to keep the house heated sufficiently to prevent the pipes from being exposed to very low temperatures. The homeowner must manually monitor this situation by increasing the temperature enough to prevent the pipes from freezing, but not too high for extended periods, as this becomes energy intensive and expensive. Another common solution is to drain all the pipes in the house. However, this approach is not a trivial task and requires professional plumbers to participate.

There are currently limited viable commercial solutions to easily monitor the temperature of the water pipes. However, even with these solutions, the user is only informed of the temperature and must then either manually increase the temperature of the home or manually drain the pipes. Thus, we developed a novel commercial product to solve this issue, while addressing the need to drain the pipes.

According to “The Freezing and Blocking of Water Pipes”, the water in pipes are subjected to the risk of freezing when the temperature is between -4°C and -6°C. Also, flow rate could be used as a gauge for the risk of pipes bursting, however there is no accurate quantitative measure for this approach[3]. Thus, our project stemmed from their findings, where we implemented a temperature sensor at the specified threshold.

Our solution actively monitors the pipe temperature, reporting such to the user via a web application interface. If the pipes are at risk of freezing, the system will automatically drain the pipes. The drainage will occur in the configured time setup by the user, which was a default to 5 seconds. The pipe status to indicate whether the pipes are operating normally or is drained is displayed on the packaged system and on the web application. The user can also command the system to drain or refill the pipes. The web application interface also plots the temperature recordings of the pipes to the user. Our solution overall provides an automatic and guaranteed solution that the pipes will never burst.

For the design expo, we constructed a pipe system to model that of a household pipe system. We demonstrated all functionality of our system. Since we could not achieve the situation where the pipes were at risk of freezing, we simulated such by placing the temperature sensor in ice water and reducing the threshold to 6°C to show the drainage process. We further showed that the pipes can be refilled by the user command on the web interface. The temperature and pipe status were also shown updating on the displays of the packaged product and the web application. We also showed that our system plots the temperature of the pipes. Finally, we showed that the Wi-Fi of our system has implemented SoftAP, in which the user can enter their SSID and password to connect the system to the internet on startup.

II. Description of project

The eventual goal of this project is to present a solution to the problem of water freezing in the plumbing system in houses during extremely cold days. More specifically, we aim to develop a commercial automatic pipe drainage system which can be relatively easy to install and work with existing pipe structures at the user's house.

The system combines sensors and traditional methodology for pipe drainage. The system involves several devices that will be installed on some specific segments of the pipes across the house. Every device serves not only as a sensor node that can measure local temperature and evaluate freezing risk, but also as the controller to some local mechanical components (valves) that will remotely collaborate to drain the pipes.

Traditional plumbing has already developed a methodology to drain a pipe. The process can be summarized in the following steps:

1. Close the main source of incoming water
2. Open all the terminals along the pipes (e.g. taps, faucets…)
3. Connect an air compressor to the faucet right next to the main entrance of water

4. Pump air into the network and push water out

Realizing that the whole process is a sequence of simple mechanical operations happening on different locations of the pipes, we believe that we can use an embedded system to do it instead. This would have been typically done by a plumber. However, it is inconvenient to request plumbing services every time the user would like to drain his pipes.

Existing research has shown that water in the pipes are subject to the risk of freezing between -4°C and -6°C[3]. Given the isolation of water in the plumbing system, temperature and flow rate are the most important factors that determines the risk of freezing. Unfortunately, the research points out that there is no good quantitative analysis on how flow rate affect water freezing.

Given the supporting evidence from the existing plumbing practice and academic research on freezing risk evaluation, we are confident that our system can achieve our goal if we can coordinate the sensors and the mechanical components to work successfully.

In order to make the product marketable, the product should be relatively small and light enough to be installed onto the pipe structure. Our packaged product was of 16cm x 10cm x 8.5 cm and weighed 0.63lbs. This was small enough to be secured onto to the pipes. However, this could have been reduced if there was time to make a second optimized version of the PCB. Another design constraint is that the cost of the system should be low enough for the user to be persuaded to purchase this product, since there would be an unavoidable plumbing cost incurred as well. Our product costs $297.96. This includes the price of all expensive components such as the valves and the air compressor. While this cost is quite high for such a system, it can be justified since it is still much less than the incurred costs of ruptured water pipes ($5,000-$50,000[2]). Next, the system should be accurate and robust, to ensure that the pipes are only drained if they are at risk of bursting. Our system has a resolution of 0.5°C, which is sufficient for this application. Additionally, our state machine used ensures that the system will always operate in a known safe state, thus preventing it from malfunctioning. Also, our system has a much faster reaction time than the rate that temperature would change. This ensures that the risk situation would be detected and dealt with before the pipe ruptures. Next, our solution should be user-friendly as it targets typical homeowners. This was achieved by designing a simple web application interface displaying only the necessary information and commands for the user. The packaged system also displayed such information. Finally, from a project point of view, we were provided a budget of $1000 and time frame of roughly 11 weeks. Our project was successfully completed within this time and kept within the budget.

Before delving into the functional requirements of our system, it needs to satisfy the following baseline requirements:

1. It must not interfere with the normal water flow in common case
2. It must not add any unsafe material to the water
3. It must not cause leakage in common case
4. It doesn't affect the structure of existing pipes

Given that our system is supposed to work in the wall, our system has two constraints:

1. It must regularly report its status to the user
2. It must interact with user via remote control

In order to make our system marketable, we want it to have the following properties:

1. It can be implemented with a small cost for the whole system
2. It can be easily installed
3. It can be configured to adapt to house with different sizes and pipe structures

There are also some important legal constraints:

1. The modification of the plumbing system must meet the Uniform Plumbing Code
2. The system must be installed and maintained by plumbers with the license

However, due to the limitation of the scope of this project. We cannot spend effort in addressing with these legal constraints.

Based on these constraints, we have categorized and defined a set of functional criterions that our system is supposed to achieve.

Table 1: Design Requirements

Category	Design Criteria	Importance	Will/ Expect/ Stretch	Achieved?	Comment
System	Drainage is triggered when any temperature is less than -4°C.	Fundamental	Will	✓	-
	The system never enters an unsafe state.	Fundamental	Will	✓	-
	Valves does not cause any leakage	Fundamental	Will	✓ (80%)	Caused by bad valve quality
	Drainage takes a constant time, which can be configured by the user through the phone.	Important	Will	✓	-
	There is a defined safe emergency mode when power and/or Wi-Fi is off.	Important	Will	✓	Redesigned so this was no longer necessary.
Web Application	Updates will take place every 10s when the user connects to web server on the mobile device.	Important	Expect	✓	-
	Push notifications occur just before pipes are drained.	Optional	Stretch	✗	This was not prioritized as a stretch goal
	Interface the app and the web server with a Nest Learning Thermostat	Optional	Stretch	✗	Did not obtain Nest Thermostat
	Record and plot temperature	Optional	Stretch	✓	Added during project
Communication	Main PCB sends data to web server when at a 10s interval when user is online or if the user is at risk.	Fundamental	Will	✓	Our system streams data to web server
	Communication occurs between main and node PCBs at 5s intervals.	Important	Expect	✓	-
	The Wi-Fi can be easily configured.	Important	Stretch	✓	-

The following diagram is an overview of the architecture of our system's circuit.

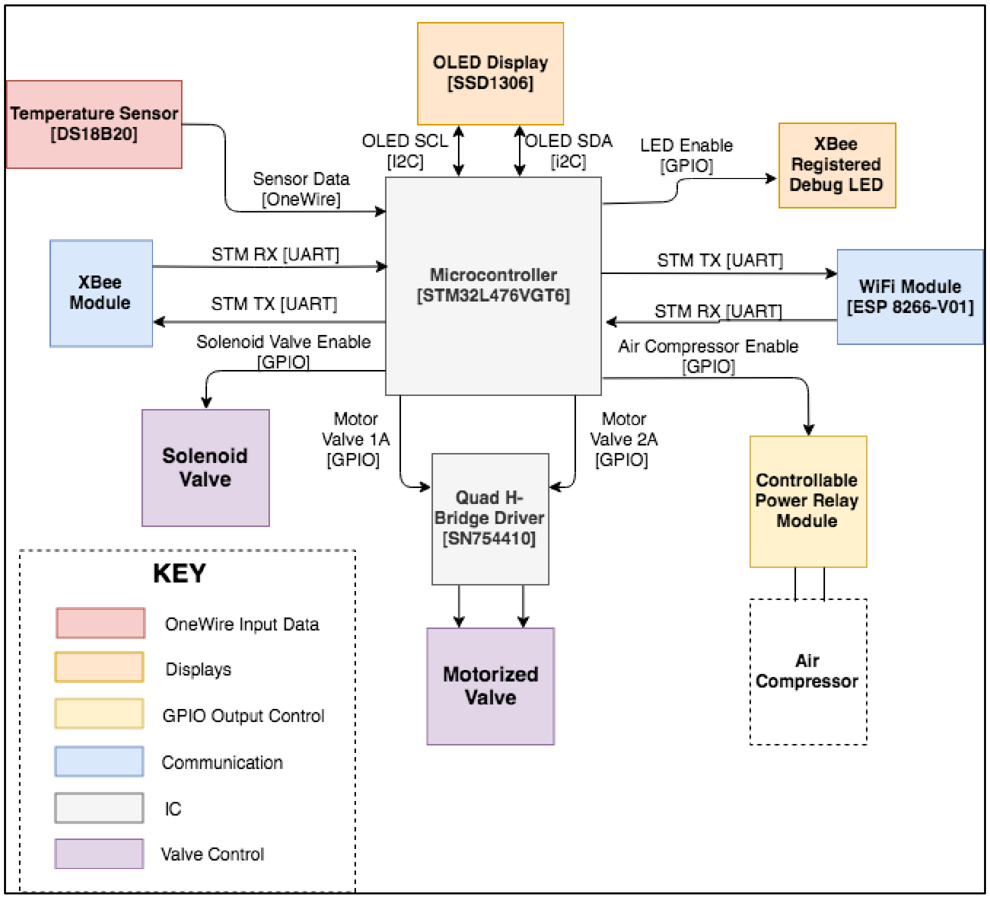


Figure 1: System Architecture

Our system involves more than one devices. One of them serves as the unique **master** device and all the other ones are **node** devices. Typically, there should be one node device for ever floor of the house. Although master and node devices share the same circuit and PCB for simplicity, they only use a subset of all the peripherals on the board.

The master device controls one motorized valve at the main water source where water enters the house's pipes at the basement. It also controls an air compressor through a power relay module which turns on the air compressor given an active high signal, and a normally off solenoid valve connecting the air compressor to the pipe.

Each node device controls two normally off solenoid valve at the terminal of pipes on each floor.

Figure 2 below shows how the master device (main PCB) and the node device is installed on an existing pipe system.

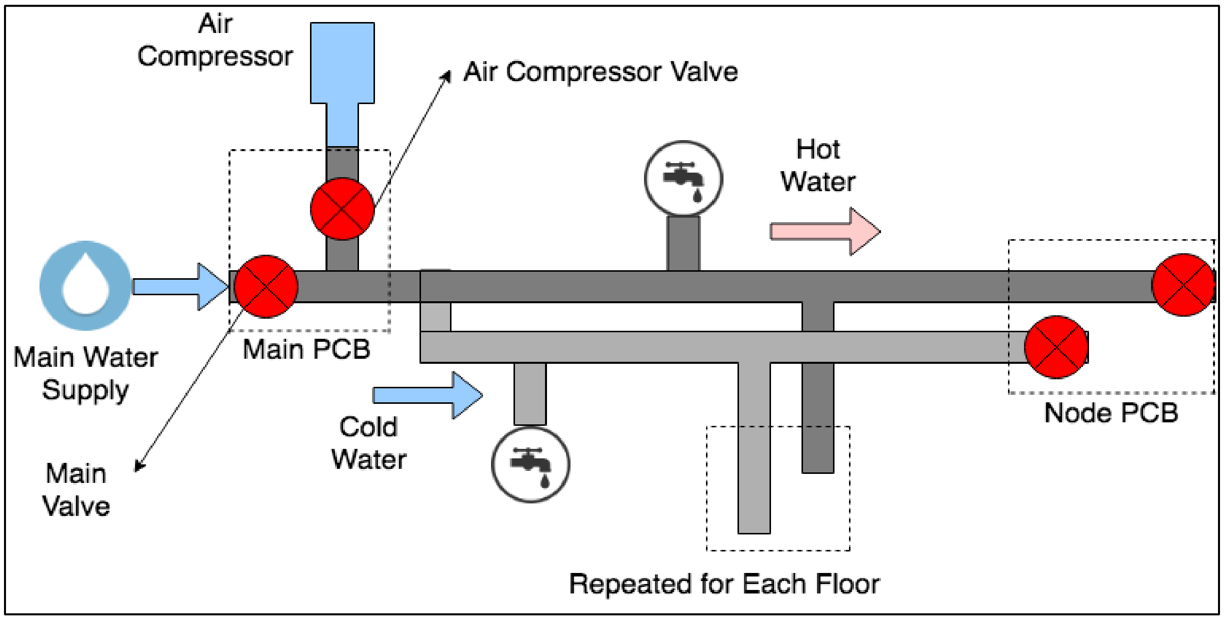


Figure 2: System Installation Layout

All the mechanical components collaborate during the drainage process, which is summarized in figure 3.

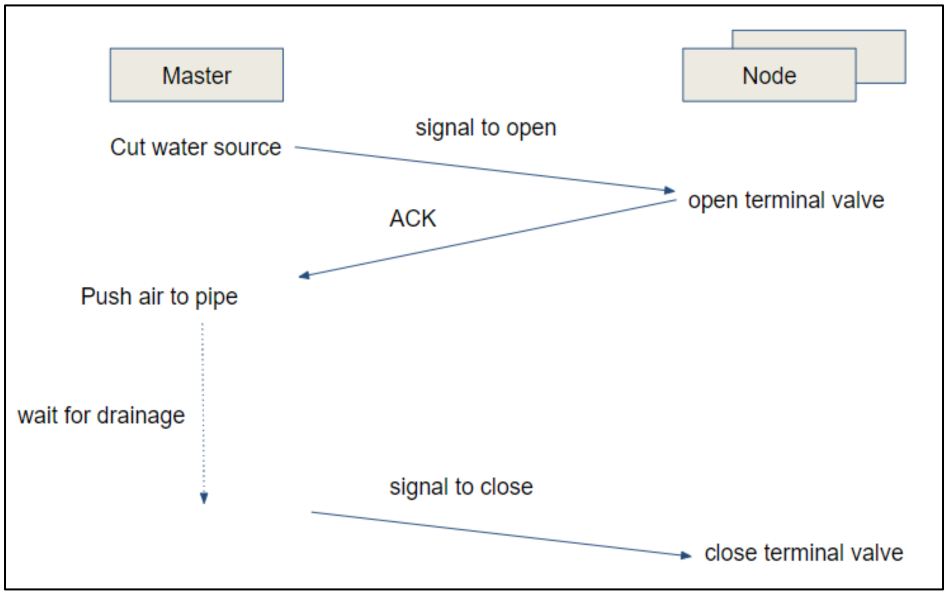


Figure 3: Drainage Process

A recover process that allows the system to recover the pipes to its normal state to start supply water is very similar to the drainage process.

A temperature sensor present on each device, collects temperature data across the locations where the devices are installed. All the temperature data are sent to the master device to evaluate the risk of freezing.

Device to device communication is handled by XBee modules. Additionally, the master device has a Wi-Fi module to connect to an external web server to indicate the state of the pipe system. It also allows the user to monitor the temperature and to it monitors to manually issue commands through the web application interface. We also gave the user the ability to configure the time taken for the pipes to drain. This can be initially setup and stored for all further drainages, or the user can also change this at any time on the web application. Furthermore, the user can view the history of the temperature of the pipes. This is shown in figure 4 below.

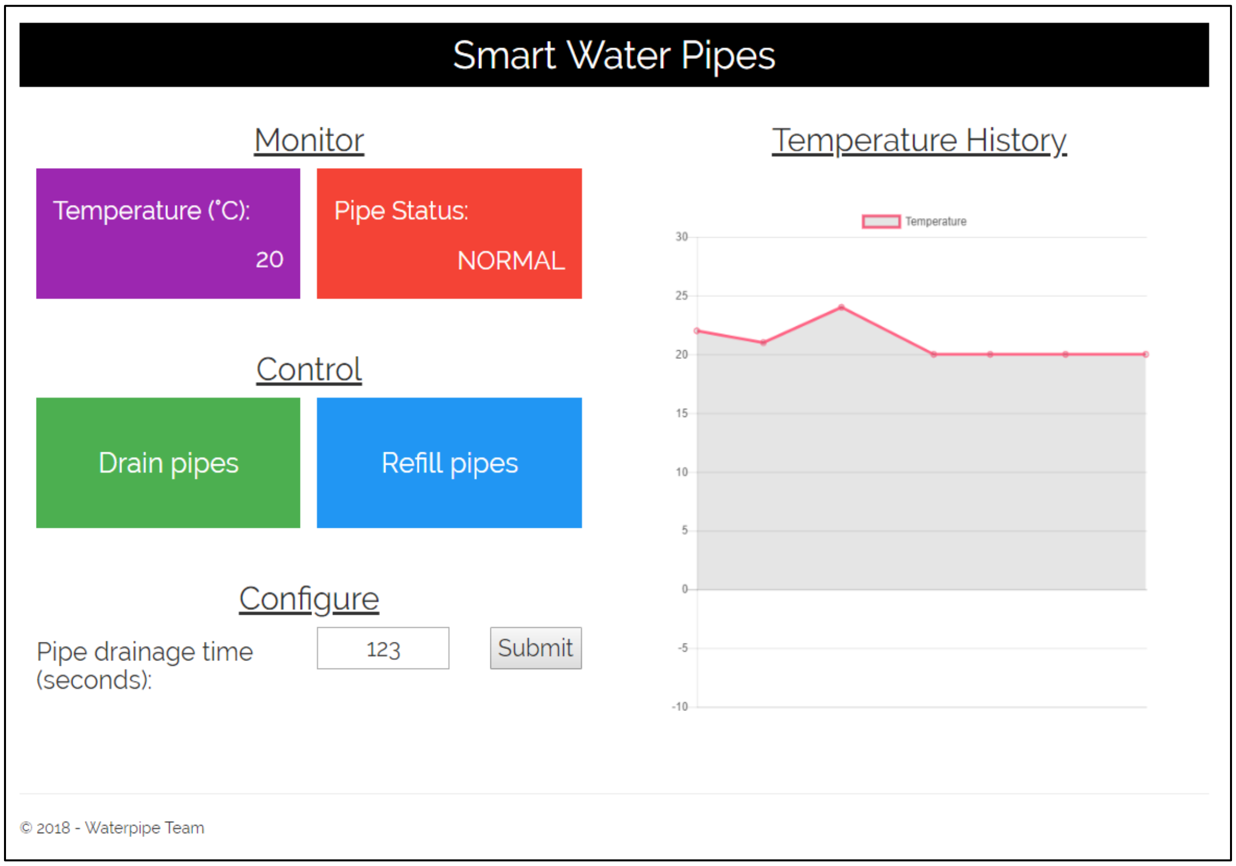
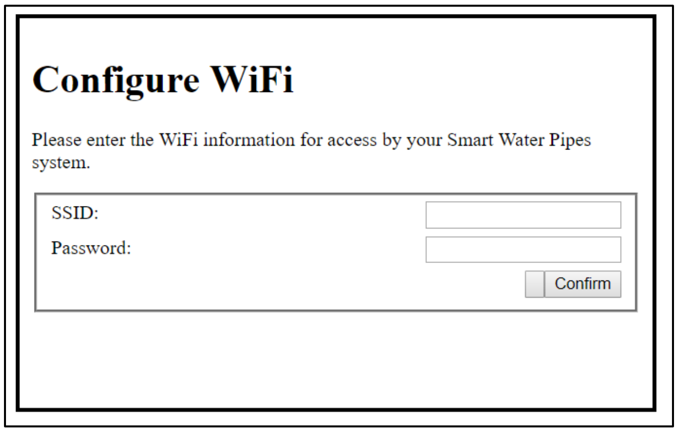


Figure 4: Web Application Interface

In order to provide a complete and simple solution, we also allow the user to configure the Wi-Fi SSID and password. This is done using the SoftAP mode of our Wi-Fi module. On startup of the system, the user will enter his Wi-Fi credentials in the interface shown in figure 5.



Configure WiFi

Please enter the WiFi information for access by your Smart Water Pipes system.

SSID:

Password:

Figure 5: Wi-Fi Configuration Interface

Every device also includes an OLED display which displays the current status of the system and local temperature measurement. An LED is added to each device as indicator of device to device communication (e.g. LED on means the device is "seeing" others). These peripherals provide necessary information for the user to understand what the system is doing. A reset button is placed on each board to restart the program on the board. After packaging our system in the box, only the OLED, LED, reset button and some holes for wires (for the mechanical components) are exposed to the user, as shown in figure 7 below.

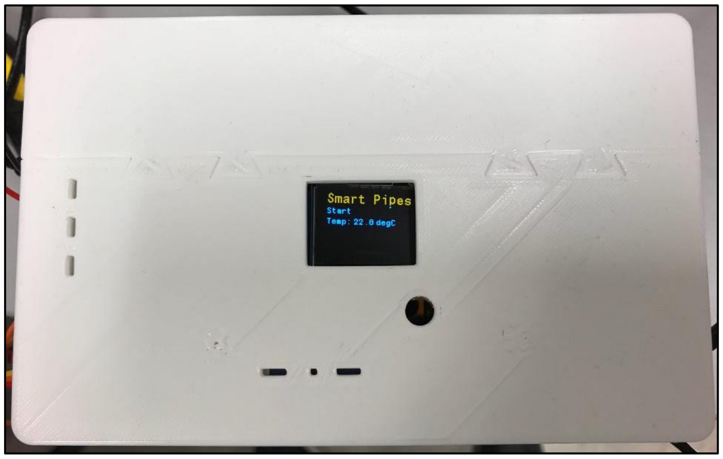


Figure 6: Packaged Product (Master Node)

III. Milestones, schedule, and budget

Milestones:

Our group successfully completed and **demonstrated** where necessary, the initial milestones. These are listed below.

Milestone 1: 10/18/2018

- **Show temperature sensor data can be displayed with development board**
- **Valves can be controlled individually with development board**
- Establish multi-node XBee communication
- Show rough draft schematic of PCBs

Milestone 2: 11/08/2018

App:

- **Show the pipe status and temperature on the web app**
- Show the current configuration on the web app
- Allow controls for user to change configurations on web server

System:

- Show overall pipe drainage process through dev board
- Respond to power outage and Wi-Fi outage on dev board
- **Display status on LCD with dev board**

Show final layout of PCBs:

- Show design of LED blink when micro has code
- Show all sensors and components layout together on PCB
- Show design of power circuits

Packaging:

- Draft of packaging for Design Expo
- Pipe structure created for demo

Schedule:

Our group was able to complete all items in our schedule by the design expo deadline. However, we did have a few minor issues along the way. Initially, we took longer than expected to select and receive our microprocessor. After such, there was a delay in getting our development environment running. This resulted in a delaying of our first major task, which was prototyping all peripherals. However, after setup, we were able to complete this step faster than planned and stuck to our schedule. Communication through XBee and Wi-Fi posed the most issues and took longer than expected. Soldering also took longer than planned, mostly due to issues faced with soldering the microprocessor. Soldering and communication tasks were done concurrently and were achieved eventually a bit delayed on schedule. The final task was system integration, final testing and improvements. We initially scheduled one and a half weeks for this. This was started a bit later due to the previous delays. However, was completed within the timeframe.

Budget:

Our group spend a total of $954.41, which was within our $1000 budget. Our proposal budgeted $661.42, which did not include some components such as the microcontroller and development board (which were not confirmed at that time) as well as passives and miscellaneous required items for the demo. Due to minor project changes, we had to also make additional purchases outside of the above, such as two motorized valves and a rechargeable li-ion battery. The PCB also costed more than initially budgeted. Overall, we were able to stick within the allocated budget.

IV. Lessons learned

Overall, our system operation went well. Our finite state machine designed was robust and thus, all our peripherals worked as expected, resulting in smooth system operation.

Our packaging was not as good as we initially planned. This item was not prioritized sufficiently and thus, we ran into difficulties with this nearing the end of the project. As a result, our final casing and overall look of our final demo setup, had a lot of room for improvement in our opinion.

If we could send a short memo back in time to advise our group, we would simply say "*Take more risks*". During our project, we took many precautionary steps. For example, with the PCB design, our design included components mainly on one side of the board, and included many additionally pins, in case we needed such. However, this resulted in a larger than necessary PCB. Similarly, we opted for a safer off the shelf solution for our battery charging system. However, ideally if we had more time, we could have designed our own.

Our group learned a wide range of technical material. Firstly, since we used a new microcontroller, we learned about a new development environment. As such, we are confident that we can now easier setup and use any given microcontroller, provided that the required documentation is available. Next, we gained a deeper understanding with XBee communication and Wi-Fi communication. Since our project implemented FreeRTOS, we gained a deeper understanding managing more tasks, as well as how to develop and maintain a good software interface. We also gained experience with building the web server and designing and building the web application. Finally, other important technical skills gained was debugging, due to the number of small issues faced during implementation. We also gained real-world engineering skills in the sense that we observed a problem, identified the current environment and issues, studied the relevant research done and engineered an applicable and efficient solution.

V. Contributions of each member of team

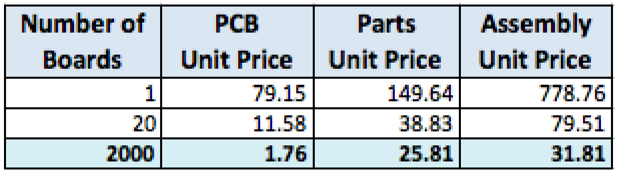
Table 2: Team Contributions

Team Member	Contribution	Effort
Rahul Gangwani	Rahul majorly developed and tested the temperature sensor. He was also solely responsible for the PCB design. He also participated in unit testing of the peripherals. He also played a large role in the packaging. Rahul also contributed greatly to the poster and documentation.	20%
Devesha Tewari	Devesha majorly developed and tested the OLED display and worked aided with the temperature sensor, Wi-Fi and web application. She soldered and tested the node PCB. She also participated in unit and system testing. She also worked a lot on the system integration and setup for the design expo. Devesha also contributed greatly to the poster and documentation.	20%
Chunan Ye	Chunan majorly developed and tested the Xbee and aided on the Wi-Fi. He also played the largest role in the algorithm development of the system with the team. He also constructed the mechanical pipe system for demo. He soldered and tested the main PCB. He also participated in unit and system testing. He also worked a lot on the system integration and setup for the design expo.	25%
Yutian Chen	Yutian majorly developed and tested the Wi-Fi. He also developed the web server side and web application interface. He also participated in unit and system testing. He also worked a lot on the system integration and setup for the design expo. Yutian also was in charge of purchasing and managing of the budget.	20%
Carl Wu	Carl aided with the development of the temperature sensor. He also did the packaging design and was responsible for the printing of the package.	15%

VI. Cost of Manufacture

By using CircuitHub, we were able to generate a quote based on the board and assembly:

Table 3: PCB Assembly Quote



Number of Boards	PCB Unit Price	Parts Unit Price	Assembly Unit Price
1	79.15	149.64	778.76
20	11.58	38.83	79.51
2000	**1.76**	**25.81**	**31.81**

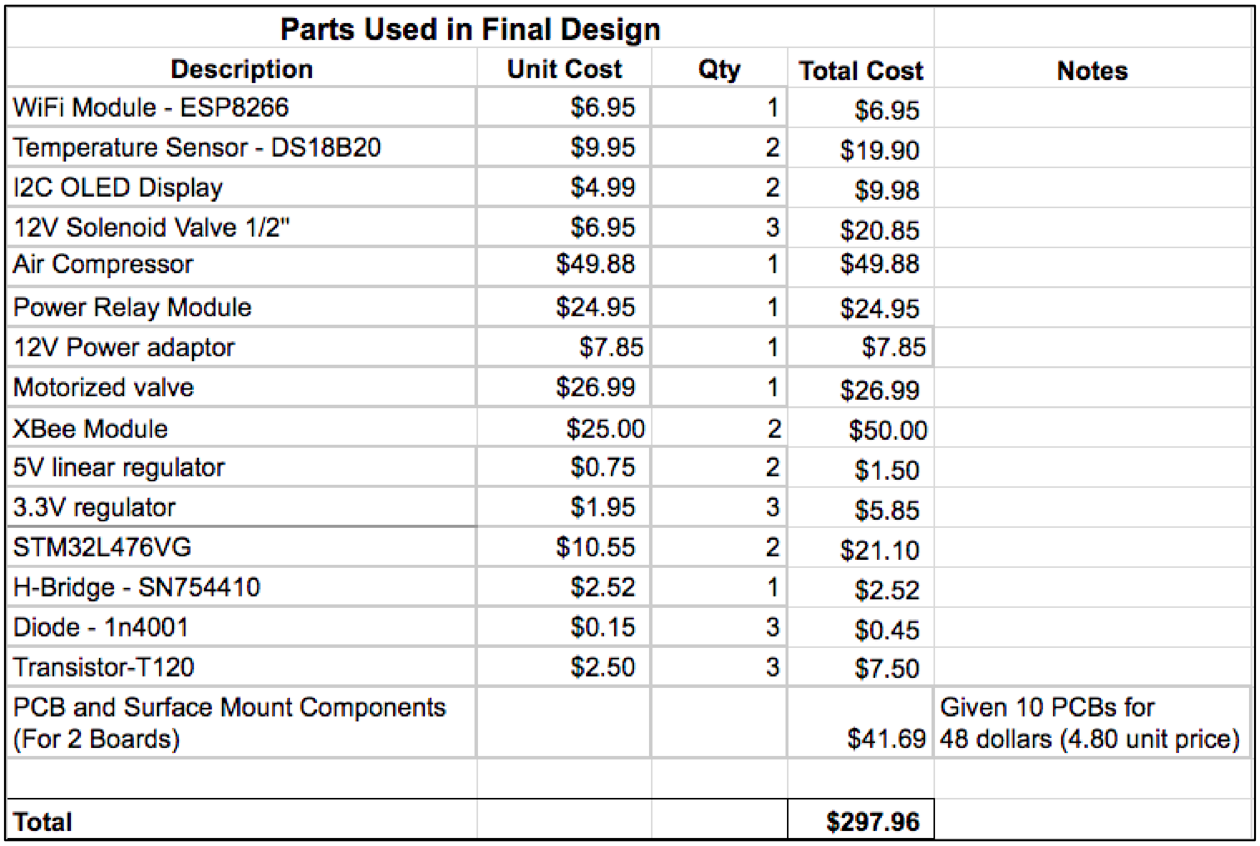
By ordering more boards at a time, we can effectively reduce the cost of the board and assembly. The reason for these prices are due to the fact that the cost of assembly is considered, and that the manufacturing process uses a pick and place machine to place components, so more of the same components are ordered to consider for attrition (parts that may be damaged by the machine).

The highlighted section of the spreadsheet shows a reasonable starting number of boards that we will need for our system. This was based on the fact that each household will have at least 4 PCBs (one for the main valve, and 3 for the node valves, one per floor). The total price per board of ordering 2000 boards with assembly is $59.38, which fits reasonably given the cost of the prototyped PCB and components without assembly was $20.84.

VII. Parts

Table 4 and table 5 below shows a list of the parts ordered that were either used in the final design or not included in the final design. These costs do not consider shipping.

Table 4: List of Parts Used in Final Design



Parts Used in Final Design				
Description	**Unit Cost**	**Qty**	**Total Cost**	**Notes**
WiFi Module - ESP8266	$6.95	1	$6.95	
Temperature Sensor - DS18B20	$9.95	2	$19.90	
I2C OLED Display	$4.99	2	$9.98	
12V Solenoid Valve 1/2"	$6.95	3	$20.85	
Air Compressor	$49.88	1	$49.88	
Power Relay Module	$24.95	1	$24.95	
12V Power adaptor	$7.85	1	$7.85	
Motorized valve	$26.99	1	$26.99	
XBee Module	$25.00	2	$50.00	
5V linear regulator	$0.75	2	$1.50	
3.3V regulator	$1.95	3	$5.85	
STM32L476VG	$10.55	2	$21.10	
H-Bridge - SN754410	$2.52	1	$2.52	
Diode - 1n4001	$0.15	3	$0.45	
Transistor-T120	$2.50	3	$7.50	
PCB and Surface Mount Components (For 2 Boards)			$41.69	Given 10 PCBs for 48 dollars (4.80 unit price)
Total			**$297.96**	

Table 5: List of Additional Unused Parts

Extra Parts Ordered			
Description	**Unit Cost**	**Qty**	**Total Cost**
Lithium Ion Battery Charger	$15.00	1	$15.00
Lithium Ion Battery - 2Ah	$12.95	3	$38.85
STM32L476 Discovery Board	$25.00	2	$50.00
npn transistor (10 pack)	$1.95	1	$1.95
PVC Pipes	$30.57	1	$30.57
32.768 kHz Crystal Oscillator	$1.89	1	$1.89
Extra STM32L476VGT6 Microcontrollers	$10.55	11	$116.05
Extra Motorized valve	$26.99	2	$53.98
Total			**$308.29**

Table 6 below shows the total cost for the two populated PCBs for the master and node device.

Table 6: List of PCB Components Used

PCB Surface Mount Components			
Description	**Unit Cost**	**Qty**	**Total Cost**
Power Jack_SMD	$1.50	5	$7.50
Schottky Diode	$0.48	5	$2.40
0.1 uF Capacitor	$0.04	40	$1.60
0.47 uF Capacitor	$0.48	2	$0.96
10 uF Capacitor	$0.24	5	$1.20
1 uF Capacitor	$0.39	1	$0.39
270 Ohm Resistor	$0.52	3	$1.56
4.7 kOhm Resistor	$0.55	2	$1.10
10 kOhm resistor	$0.37	10	$3.70
SPST Tactile Push Button	$6.98	1	$6.98
680 Ohm Resistor	$0.01	2	$0.02
LED Red	$0.06	2	$0.12
LED Green	$0.06	2	$0.12
LED Yellow	$1.90	2	$3.80
22 Ohm Resistor	$0.03	8	$0.24
330 Ohm Resistor	$0.10	2	$0.20
510 Ohm Resistor	$0.10	2	$0.20
PCB Boards	$4.80	2	$9.60
Total			**$41.69**

VIII. References and citations

Interface Code:

The main header files written for our project are shown below.

```
/*Temperature.h*/
#ifndef TEMPERATURE_H_
#define TEMPERATURE_H_

#include "stm32l4xx_hal.h"

#define ONEWIRE_CMD_SKIPROM                 0xCC
#define ONEWIRE_CMD_RSCRATCHPAD             0xBE
#define ONEWIRE_CMD_WSCRATCHPAD             0x4E
#define ONEWIRE_CMD_CPYSCRATCHPAD           0x48
#define TEMP_CMD_CONVERTTEMP                0x44     /* Convert temperature */

#define HIGH 1
#define LOW 0

typedef enum Temp_Command
{
    READ_TEMP, SET_RESOLUTION, CONVERT
}
Command;

typedef struct Temp_Struct
{
    GPIO_TypeDef* GPIOx;            /*!< GPIOx port to be used for I/O functions */
    uint16_t GPIO_Pin;              /*!< GPIO Pin to be used for I/O functions */
}
Temperature;


/* HIGH LEVEL FUNCTIONS */
void initTemp(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, GPIO_TypeDef* port, uint16_t data_pin); // Initialize the instance with a GPIO pin
float getTemperature(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst); // Read temperature data from the sensor
double convertCelsius(double fahrenheit_in); // Convert Fahrenheit to Cenlsius
double convertFahrenheit(double celsius_in); // Convert Cenlsius to Fahrenheit


/*LOW LEVEL FUNCTIONS */

//Setup pin to use OneWire protocol through temp Init
void OneWire_Init(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin_in);

//Step 1. Initialization - Send pulse and wait for response, if good move on.
uint8_t OneWire_Reset(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst);

//Step 2. ROM Command -> Skip ROM
```

```
void Temperature_SendCommand(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, Command cmd);

//Set R0 and R1 to 0 for 9 bit resolution
uint8_t Temperature_SetResolution(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, const uint8_t resolution);

////Write Logic
void OneWire_WriteByte(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, uint8_t byte);
void OneWire_WriteBit(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst, uint8_t bit);

////Read Logic
uint8_t OneWire_ReadBit(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst);
uint8_t OneWire_ReadByte(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst);

//Temperature convert and read result logic
void Temperature_Convert(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst);
float Temperature_Read(Temperature* TempPin, TIM_HandleTypeDef* Tim_Inst);  //Skip ROM start temperature conversion

//Invalid command was entered
void Invalid_Cmd();

//OneWire Pin Manipulations
void SET_ONEWIRE_LOW(Temperature* TempPin);
void SET_ONEWIRE_AS_OUTPUT(Temperature* TempPin);
void SET_ONEWIRE_AS_INPUT(Temperature* TempPin);
void SET_ONEWIRE_DELAY(TIM_HandleTypeDef* Tim_Inst, uint32_t time_us);
#endif /* TEMPERATURE_H_ */
```

```
/*Valve.h*/

#ifndef VALVE_H_
#define VALVE_H_

#include "stm32l4xx_hal.h"
#define HIGH 1
#define LOW 0

//========= Solenoid valve. One-pin control

// Solenoid valve status
typedef enum valveSoleStatus {
    OPEN, CLOSE
} ValveSoleStatus;

// Solenoid valve struct
typedef struct valveSole {
    GPIO_TypeDef* GPIOx;
    uint16_t pin_0;
    ValveSoleStatus status;
} ValveSole;

// Solenoid valve high level functions
```

```
void valveSoleInit (ValveSole *valve, GPIO_TypeDef* GPIOx, uint16_t pin_0); // Initialize with 1 GPIO pin
void valveSoleOpen (ValveSole *valve); // Open the valve
void valveSoleClose (ValveSole *valve);// Close the valve


//========== Motorized Valve. Two-pin control

typedef enum valveMotorStatus {
    OPENED, CLOSED, OPENING, CLOSING
} ValveMotorStatus;

// Motorized valve struct
typedef struct valveMotor {
    GPIO_TypeDef* GPIOx;
    uint16_t pin_0;
    uint16_t pin_1;
    ValveMotorStatus status;

} ValveMotor;

// Motorized valve high level functions
void valveMotorInit (ValveMotor *valve, GPIO_TypeDef* GPIOx, uint16_t pin_0, uint16_t pin_1);// Initialize with 2 GPIO pins
void valveMotorOpen (ValveMotor *valve); // start opening the valve
void valveMotorClose (ValveMotor *valve); //  start closing the valve
void valveMotorHold (ValveMotor *valve); // stop the valve


// Valve instances
ValveMotor mvalve1;
ValveSole svalve1;
ValveSole svalve2;
ValveSole svalvee;

#endif /*VALVE_H_*/
```

```
/*SSD1306.h - OLED*/
#ifndef __SSD1306_H__
#define __SSD1306_H__

#include "stm32l4xx_hal.h"
#include "ssd1306_fonts.h"

/* I2C config*/
I2C_HandleTypeDef hi2c1;
#define SSD1306_I2C_PORT        hi2c1
#define SSD1306_I2C_ADDR        (0x3C << 1)

// SSD1306 OLED height in pixels
#define SSD1306_HEIGHT          64
// SSD1306 width in pixels
#define SSD1306_WIDTH           128

// Enumeration for screen colors
```

```c
typedef enum {
    Black = 0x00, // Black color, no pixel
    White = 0x01  // Pixel is set. Color depends on OLED
} SSD1306_COLOR;

// Struct to store transformations
typedef struct {
    uint16_t CurrentX;
    uint16_t CurrentY;
    uint8_t Inverted;
    uint8_t Initialized;
} SSD1306_t;

// High-Level functions
void ssd1306_Init(void);
void ssd1306_Fill(SSD1306_COLOR color);
void ssd1306_UpdateScreen(void);
void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color);
char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color);
char ssd1306_WriteString(char* str, FontDef Font, SSD1306_COLOR color);
void ssd1306_SetCursor(uint8_t x, uint8_t y);

// Low-level functions
void ssd1306_Reset(void);
void ssd1306_WriteCommand(uint8_t byte);
void ssd1306_WriteData(uint8_t* buffer, size_t buff_size);

#endif // __SSD1306_H__
```

```c
/*Xbee.h*/

#ifndef XBEE_H_
#define XBEE_H_

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include "stm32l4xx_hal.h"
#include "usart.h"
#include "gpio.h"

#define TRANSMIT_TIMEOUT 200
#define RECEIVE_TIMEOUT 300
#define RECEIVE_BUF_SIZE 10
#define MAX_SLAVE_NUM 3

// message type
#define SYNC 0
#define NEW_ID 1
#define REG 2
#define DATA 3

```

```c
// command type
#define NO 0 //No action
#define DRAIN 1 //Drain action
#define RECOVER 2 //Recover action
#define END 3 //End action

// check slave alive period - 10 cycles
#define REPORT_PERIOD 10

typedef enum {
    START,
    NORMAL,
    DRAINING,
    DRAINED,
    RECOVERING
} XbeeState;

typedef struct
{
    // UART
    UART_HandleTypeDef *husart1;
    // generic
    char buffer_rx[2];
    int id;
    uint8_t need_action;    // From temp task and xbee task
    uint8_t need_recover;   // From WiFi task
    // master
    uint8_t slave_vector[3];
    uint8_t slave_report_vector[3];
    uint8_t responded_slaves;
    XbeeState state;
    uint8_t report_round;
} Xbee_t;

extern Xbee_t Xbee;

// High level functions
void xbee_init(UART_HandleTypeDef *husart, int xbee_id); // initialize master Xbee with id
 and UART port
void send_data(int is_ACK, int need_action); // client xbee send command
void send_sync(int action); // master xbee send command
void send_new_id(int new_id); // master send an id to clents
void send_reg();


// Internal functions
bool xbee_send(char b); // send raw data
void Xbee_RxCallBack();

#endif /* XBEE_H_ */
```

```c
/*WiFi.h*/

#ifndef WIFI_H_
#define WIFI_H_
```

```c

#include "FreeRTOS.h"
#include "task.h"
#include "cmsis_os.h"

#include "stm32l4xx_hal.h"

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>

// WiFi status
typedef enum{
    SOFTAP,
    STATION
}WiFiMode;

// WiFi Struct
typedef struct{
    // public variables
    WiFiMode mode; // wifi status
    bool IsNewMessage; // If there is a new message from wifi module
    char receive_state[4]; // Command parsed from a message
    char config_time[6]; // Configuration time parsed from a message

    // private variables for data transmission
    char rxbuf;
    char messagebuf[15];
    char databuf[15];
    int bufindex;
    int data_lenth;
    UART_HandleTypeDef *husart;
}WiFi;

// WiFi instance
WiFi wifi;


// Wifi high level functions
void wifi_init(UART_HandleTypeDef *husart); // Initialize the instance with a UART port
bool wifi_send_data(int temp, char* status);// Send temperarture and system status to WiFi
 module
bool wifi_get_data(); // Read data from the receive buffer
void wifi_callback(); // UART callback handler


// INTERNAL FUNCTIONS
bool wifi_checkmode(); // check the mode of WiFi
bool wifi_message_check(); // check whether there is a new message

#endif /* WIFI_H_ */
```

<u>Soldered PCB:</u>

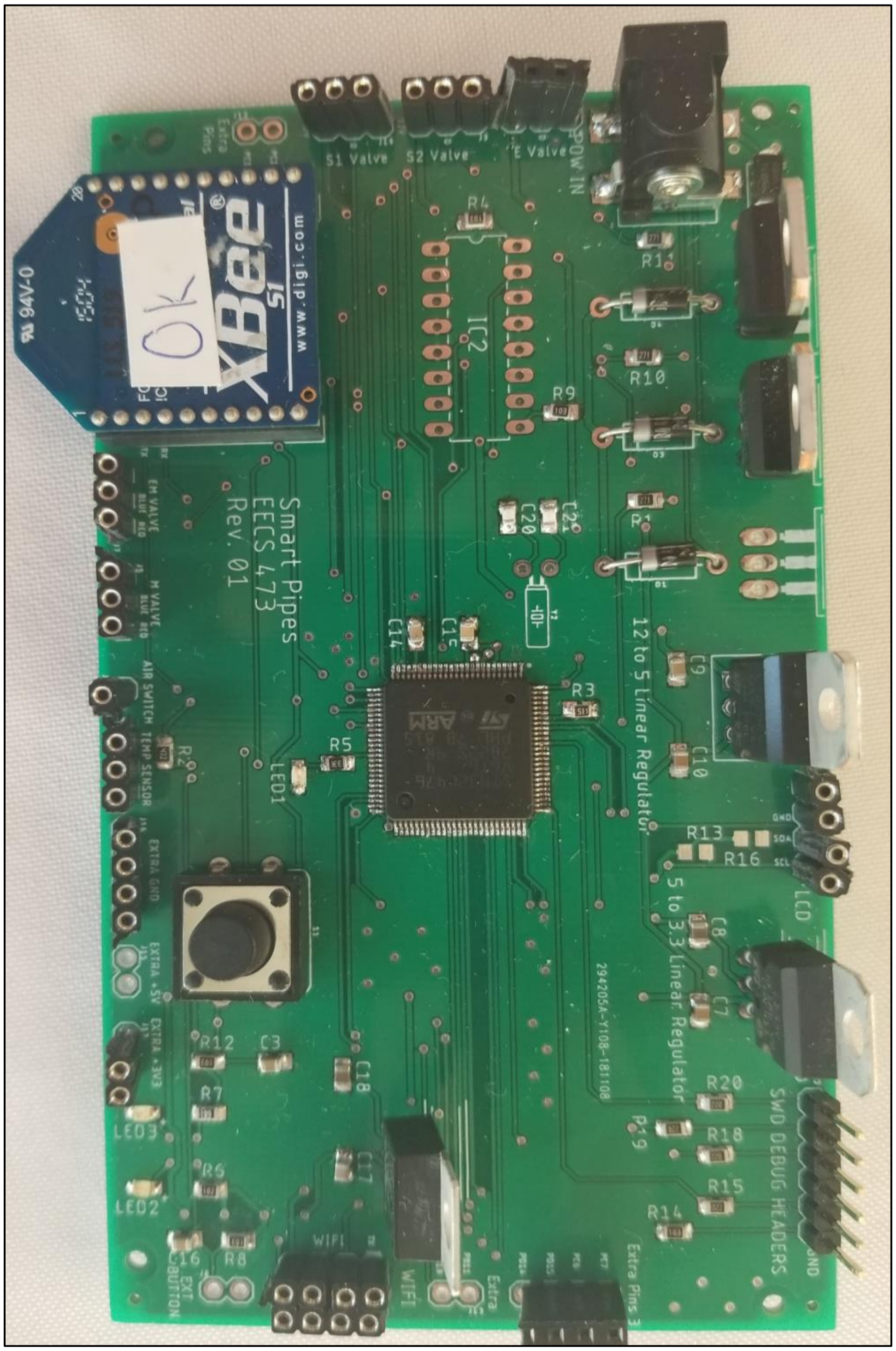


Figure 7: Front Side of Soldered PCB

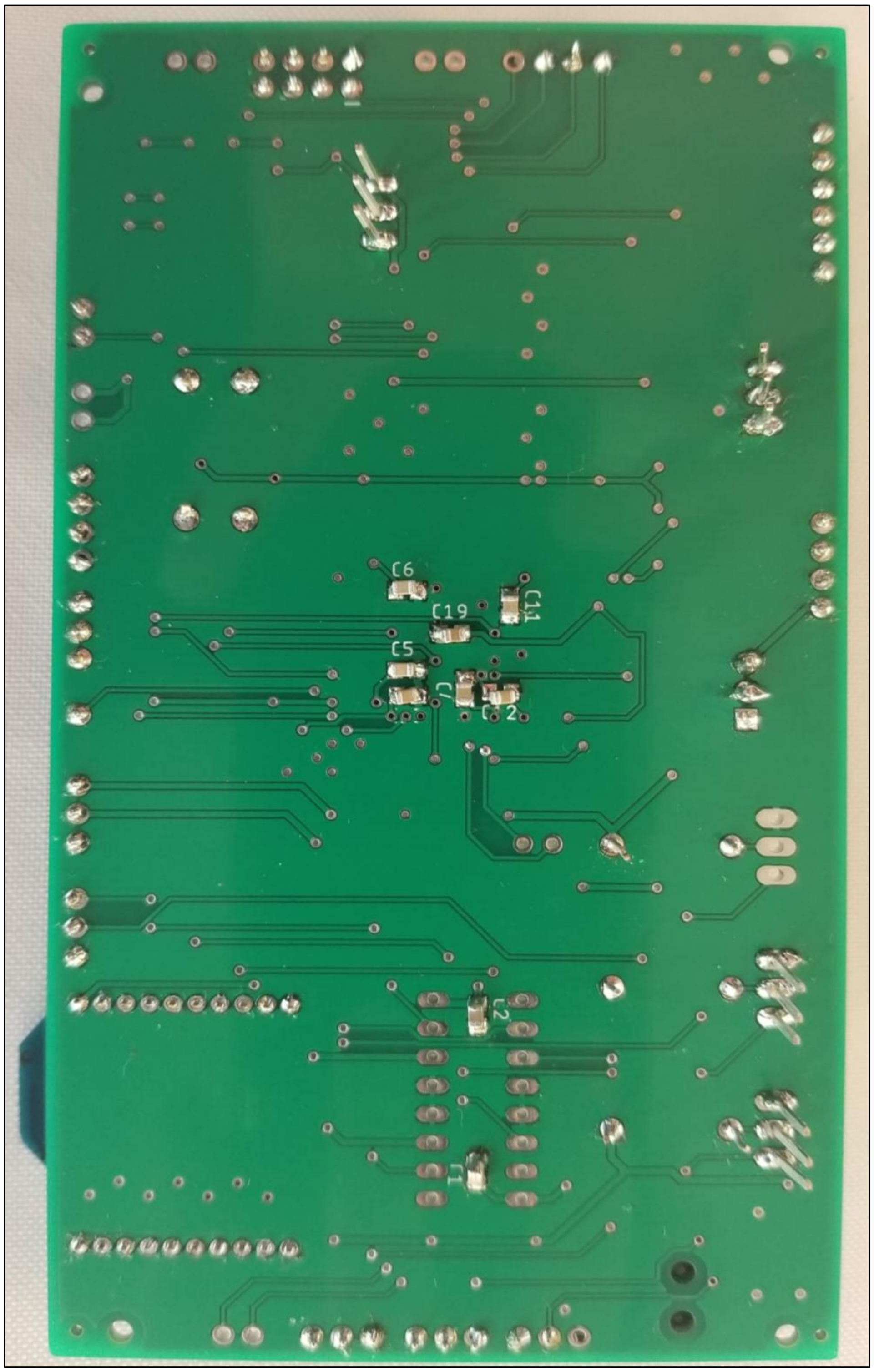


Figure 8: Back Side of Soldered PCB

Open Source Resources Used:

The following open source code resources were used:

1. C Standard Library was used. This includes:
   - stdio.h
   - stdlib.h
   - string.h

2. STM32 Library was used. This includes:
   - FreeRTOS
   - STM32_HAL

3. Arduino Library was used for the ESP8266 Wi-Fi module. This includes:
   - FS.h
   - PString.h
   - Streaming.h
   - ESP8266WiFi.h
   - DNSServer.h
   - ESP8266WebServer.h
   - ESP8266HTTPClient.h

4. Additional code resources include:
   - Tutorial for hosting a webpage on ESP8266 - https://yoursunny.com/t/2017/freewifi/
   - Source code for SSD1306.h - https://github.com/afiskon/stm32-ssd1306

References:

1. MSP Divisions Emergency Management & Homeland Security Winter Emergency Preparedness. (n.d.). Retrieved from http://www.michigan.gov/msp/0,4643,7-123-72297_60152_70432-342721--,00.html

2. How Much Does a Burst Pipe Cost? - CostHelper.com. (n.d.). Retrieved from https://home.costhelper.com/water-leak.html

3. Carey, K. L. (1892, May). The Freezing and Blocking of Water Pipes. Cold Regions Technical Digest. Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/a148943.pdf