COMP/EMEC 462 Embedded Systems

Final Project Report

(Fall, 2021)

Rainfall Inches Measuring Machine (RIMM)

Department of Computer Science

California State University, Channel Islands

Prepared by

Daniel Rojas

I. Problem Statement

Over the recent years I have seen less and less rainfall occur in Santa Barbara County, CA. I remember when I was in elementary school how much more often and harder it would rain in the city of Santa barbara and Goleta. However as time has gone on, my favorite weather seems further away from this county's grasp. Therefore, I decided that the first fundamental step to resolve this environmental issue would be to first measure the amount of rainfall year-round in various parts of the county. This is where this project comes in. Favorably, this machine would be placed in a multitude of uncovered areas where rainfall would reach it. By having many spread out over a wide area, we can get a better grasp on which areas are more affected by droughts or too much rainfall. This data could be used to assist prevention efforts for wildfires or hillside structural collapse.

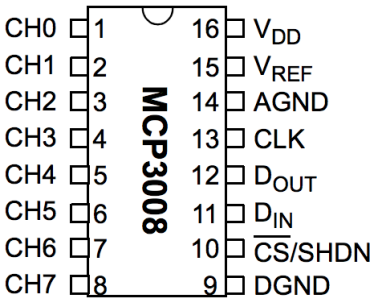
II. Proposed Solution

The solution to the aforementioned problem, and the goal of this project was to develop a web interface integrated system which would deploy an analog water level sensor, which would be interfaced by a SBC (Single-Board Computer). In our case, a Raspberry Pi. Although we could have used a MCU (Microcontroller Unit) such as an Arduino, I decided that it was better to use an ADC Microchip that could act as a bridge between the digital Raspberry Pi, and the analog sensor via a SPI connection, rather than having to connect a Single Board Computer and a Microcontroller together. It seemed like overkill to me, and a waste of space if we were to ever reach the development stage where we would design casing for the machine. The objective of this system is to build a machine which would measure the amount of rainfall in an hour. This machine would stay running as long as needed. I originally planned for it to include a motor which would pour out the water in it's rain collector every hour. However this has not been implemented as a part of the system yet, if this project were to continue this would be the next step. With all of this into consideration, my system only contained three main components besides wiring: Raspberry Pi 3B+, MCP3008 ADC Chip, and ELEEGO Water Level Sensor. As of now I am using a breadboard for prototyping, but when deployed the device would preferably be soldered together using a more cost-effective computer board such as a Raspberry Pi Zero W. On a different note, the web interface would include the following pages: A home page, a graph page, a chart page, a live page and a table page. Their functionality is described more in depth in the Development section.

III. Development

Initially my first priority was learning how the sensor I chose worked. It took me several days until I was able to successfully write software which utilized the Analog to Digital Converter to communicate with the RPi and the sensor via an SPI connection. To communicate via SPI was tricky mainly due to the fact that I had to

perform bit operations in order to not only read data from the ADC, but also do so when writing data back. I accomplished this using the RPi GPIO Library for Python, which enabled me to set some GPIO pins as input, while setting others as output. The ADC Chip interprets a high signal as a logical bit one, and a low signal as a logical bit zero. Speaking of GPIO pins, here are the RPi pins I used and where they connect to the ADC Chip: GPIO10 = Dout, GPIO9 = Din, GPIO11 = CLK, GPIO8 = CS, 3.3v = VDD and VREF, GND = AGND and DGND. The water level sensor must connect to the chip as follows: + = VDD or VREF, - = AGND or DGND, S = CH0. This is all according to the following chart,



The web interface was developed using Flask through Python. This technology allowed me to connect my Python script and pass in data to any HTML files, which would then be displayed as a local web server on port 5000. Furthermore, I utilized Jinja, which came bundled with Flask. Jinja allowed me to work with python objects within the HTML files and it also let me embed Javascript scripts into HTML files in order to be able to display a chart and a graph of simulated monthly data stored in a csv file. Apart from those two visual interpretations, the server has a page dedicated to the most recent live reading of the sensor, plus another page with a table displaying all the data collected for the current day as of the time it was accessed. Finally, the most important page is the home page, with which we would use to access all these other pages through the use of embedded links. In other words, it all combined acts like a simple website to visualize our data.

As a result of choosing to use Flask, I encountered a difficult problem when interconnecting the script which ran the server and the script which interfaced with the sensor. In order for my server to function as intended it needed to be able to receive the data read by the sensor script. This was one of the most tedious issues I had to deal with in the entire project. I tried many things, using system environment variables, starting one process from another and sharing global variables, storing the sensor script as a method and executing it as a different thread from the server and sharing memory, etc... After various trials and errors, I ultimately reached my solution. I used the multiprocessing python module. This module let me start the server script as the main program , and within it I created a queue which would let me share information with another process as long as it too had the exact same queue passed in as an argument. This meant that I encapsulated my sensor script as a method which would run infinitely with the use of an infinite while loop and a delay. I passed in the queue by calling the method from my server script as a child

process in the background, which would let me run both scripts at the same time. Whenever one script had to read or write to a data file, it would not freeze the other, rather it would continue independently, sharing or receiving data through the queue when needed.

Another big issue I encountered was how unreliable, difficult and inconsistent the water level sensor was to work with. One of the biggest issues with it's design is that it is more apt at recording moisture in the atmosphere continuously rather than coming into contact with liquid water directly. The problem is that once the water level recedes by removing water from the container, there is still a film of water covering the sensor due to the surface tension of water. Meaning that it would incorrectly record a portion of the water that was not there anymore. For it to dry in a timely manner when it would be raining would be next to impossible due to the moisture in the air. All around I made quite a bad decision in choosing to work with this sensor. However, I do not regret it. I still wanted to proceed with my project because I felt passionate about what its objective was. I did learn throughout the process that a sonar sensor would have been far better for this application, because we could bounce ultrasonic waves off of the water surface and calculate the distance compared to an empty container. I was considering a change to the sonar sensor, unfortunately we had already used it in a previous lab. Making the change would contradict the project requirements. Thus I decided to proceed with the water level sensor.

IV. Conclusion

Overall, I thought that this was a great learning experience, not only through the success I found, but also through the failure I endured. I was too optimistic about my time frame when writing my initial proposal, it really put into perspective the work that goes into the creation of more complex embedded systems. I learned that the product is never finished, there is always something more that you want to implement or an old idea that you want to improve upon. In the future if this project were to continue, then the next logical step as I mentioned before would be to implement a motor-container pair that would pour the water out by itself using a python script. I would need a motor driver for this too. Furthermore, I would definitely change from the water level sensor to the sonar sensor. I would also implement the other features I originally had in mind, which include things such as water-proof cursing and automatic foliage protection/cleaning for the container. In a bigger picture once I implement these changes, then I would shrink down the form factor while at the same time reducing the cost of the machine so that I can build multiple of them and spread them out over various locations. If this does happen then I would also need to deploy my web server using nginx and employ port forwarding to allow WAN access. Furthermore, I would move the web server to a dedicated machine so that more than one machine could be connected to the web interface running on a central hub. I do recognize this is easier said than done, but it would be a great learning experience if I ever do decide to finish implementing my ideas.

REFERENCES

MCP3008 ADC Datasheet:

https://datasheet.octopart.com/MCP3008-I/P-Microchip-datasheet-8326659.pdf

APPENDICES

Project Code Repository (Includes README with instructions on how to run):

https://github.com/DRojasCSUCI/RIMM_Rainfall-Inches-Measuring-Machine