Aquaponic Data Collection System

Final Report for

Ed Casas *(ELEX 4560)*, Bob Gill *(ELEX 4560)*, Susan Woo *(Comm 2443)*

Prepared by

Jeffrey Batac, Mateusz Nieścierowicz, James Solibio

December 3, 2018

Table of Contents

Contents

List of Illustrations

Figures

Tables

Summary

An SFU mechatronics student is developing a fully automated aquaponic system. He gave the idea to our team to prototype a data collection system that would send the data over Wi-Fi to a web server where it could be accessed by any device. The system would monitor the data, and any value deemed out of range would have an action associated with it. The entire system would have to be secured to prevent random users from connecting to the website and using it for other purposes.

Our system can execute the desired system of the SFU student with some minor changes. In our original proposal, we said we would have the web server displaying the data pulled from the data collection system and having it graph it simultaneously. It proved to be too ambitious, and we had to cut that and instead make use of Google's free developer services. With the use of Google API, we created a secure way of storing the collected data by protecting it behind a Google account. We used Google spreadsheets to store the data and to send notifications to the user about out of range values.

Working with the SFU student also helped with our primary goal of keeping the system costs as inexpensive as possible. The overall budget of the project was $200. Our initial build was a total of $136.88, but we were able to cut much of cost due to borrowing or receiving free parts for the project. The SFU student was also able to reduce our costs down with solutions that did not require parts for. Our overall cost for this project was $23.80, which were the two sensors.

The schedule was proven to be very tight, and there were many times where the team had to double down on the work in order to meet the quota's set out at the beginning of development. In the end, we were still able to finish a week early thanks to the development of an alternative solution from the SFU student.

In conclusion, the data collection system works the way the prototype was designed to work. Although we had to change how the data was being stored, the Google developer solution still proved to be enough for the prototypes needs. Finally, the primary focus of keeping the system cost as economical as possible was met.

The team highly recommends that in the future a bigger emphasis on the script that is running the collection of data. Currently, there are many flaws that either stop the process of collecting data or hinders one of the objectives of the data collection system.

Introduction

The purpose of this project was to demonstrate the team's knowledge and skills, acquired from BCIT's Electrical and Computer Engineering Technology: Telecommunications and Networking Diploma program. The project also serves as a prerequisite for graduation from BCIT.

The data collection system is a part of a much larger project for an SFU mechatronic systems capstone project. The SFU project is a fully automated aquaponic system. A colleague of Jeffrey Batac granted him permission to develop a preliminary system that will be further developed later. Jeffrey acted as proxy the for the SFU student, relaying information from the student to the team about design changes and project direction.

An aquaponic system is a type of garden that utilizes the inhabitants of a natural ecosystem to grow and harvest crops and fish. Fish are present in the system to produce waste. The waste they excrete is then converted into fertilizer by micro-organisms in the water. The byproduct they produce will provide the plants with the nutrient that they are missing from the soil. The plants filter the water that return to the fish [1].

This final report is prepared for Susan Woo – the instructor for COMM 2443, Bob Gill – the project mentor, and Ed Casas – Program head and instructor for Telecom & Networks Projects (ELEX 4560). Our final report will show our findings over the last few weeks.

The final report contains the project overview, the cost of supplies, the original and revised schedule, the team's conclusions, recommendations, and an appendix containing extra information about the project.

Project Overview

The main idea behind the fully aquaponic garden was to reduce the amount of time that the owner, of the garden, would have to maintain the system. Ideally, the only time the owner would have to check in on the garden was to collect produce or if a sensor had stopped working. However, the occurrence of a sensor going down would be very low, but the design was to consider that aspect as well.

Design

The SFU student gave us free rein to design the system to the best of our abilities; however, he specified that the system must send the data over Wi-Fi to a web server and the actions associated with the system must control the environment. Our initial design looked much like Figure 1, we had intended to use three sensors and have two actions associated with the sensors. One was going to be a fan or LED, and the other was the water pump which would regulate the amount of water in the growth bed or fish tank.

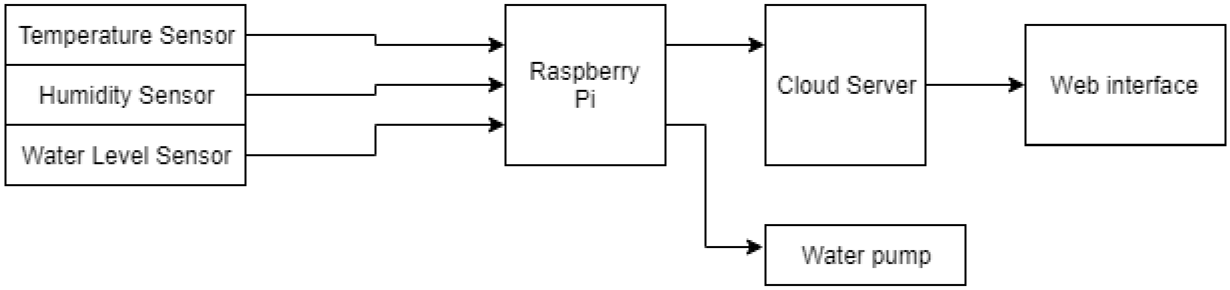


Figure 1: Block Diagram of proposed Data Collection System

However, the SFU student came up with a solution which eliminated the need for the water level sensor and water pump. He came up with a bell siphon (See Appendix A: Bell Siphon) which would automatically regulate the water that was present in the system. The water in the growth bed would constantly be flushed out if it became too full. The water level in the fish tank would be constant since the bell siphon would continuously drain the water from the growth bed into the tank. With the bell siphon now regulating water level the new block diagram was much simpler, and now the only action was an LED.

Another design change that was made was how we were going to store data. This change was not influenced by the SFU student but was a time sensitive change. The original design had us creating a website where it would display the current data values of the system and have a trend graph of the past several hours. The idea was too ambitious for the time we had so we opted to use Google developer API.

The solution at first seemed perfect, the Google developer API was free, Google Drive was secure, and Google would be backing up all the files. In hindsight, the solution was very limited and there wasn't much we could do about how the data was being handled. It also became a very complex way of storing data which would hinder the user-friendly setup. However in retrospect, it is a proof of concept that the raspberry pi can take the data and store it somewhere online.

The Google API solution also created a secure way of storing the data since the spreadsheet where the data was being stored in was in a Google account. The Google account cannot be accessed unless the person entering the account has their phone number connected to the account.

The connection to the Google spreadsheet was to be done by Wi-Fi. For this, we decided to use a raspberry pi 3 model B which had built-in Wi-Fi. However, we knew that raspberry pi's did not like the security of BCIT's network, so we had to use a hotspot from one of our phones to provide Wi-Fi access.

Operation

Raspberry Pi Side

We designed the system to start autonomously as soon as the raspberry pi was given power. We accomplished this process by implementing, an *'autostart'* script into the startup configuration of the pi. The *'autostart'* script would open on start up of the raspberry pi and then open the command terminal. Once the command terminal was opened, the script would then navigate the files to find another script called *'start'* which would just navigate the files again to start the main script.

After the main script was opened, the program would then check to see if the JSON file was correct. If the correct credentials were given to the main script, the raspberry pi would be given access to the google account. If the JSON file was invalid, the raspberry pi would not start collecting data. Figure 2 shows the operation of the raspberry pi once the script has been started. As mentioned, if the pi does not gain access to the Google account, it will prompt the user that it could not reach the account, and the JSON file must be changed (See Appendix B: .

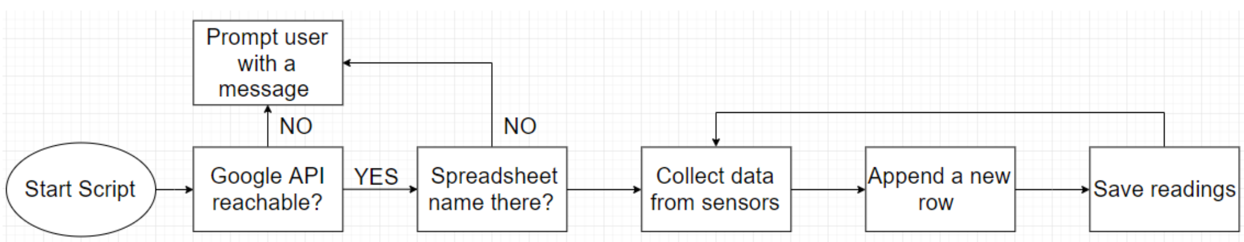


Figure 2: Flow Chart of the Data Collection System

Once the script is granted access to the spreadsheet, it will check to see if the spreadsheet it wants to enter data into is there. Once again if the script cannot find the spreadsheet, the code gets must be modified for the program to work. If the spreadsheet is there, the script will start collecting data from the sensors. The script appends rows to the Google spreadsheet, and it will infinitely continue to do so until the spreadsheet becomes too big or if the power to the raspberry pi is cut (see Appendix B.3: Main Script Code.

The script will also check the readings of the data being collected; Figure 3 shows the operation of the actions which are associated with the data that is being sent to the spreadsheet.

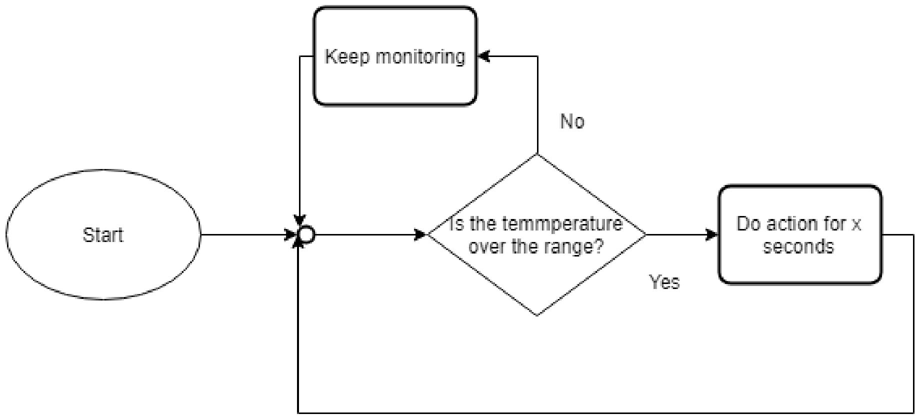


Figure 3: Flowchart for actions based on data readings

The operation is simple; the script will continuously compare the last reading to a value which can be set by the owner. Once the script detects that the last value it read was out of range, an action would begin depending on which sensor was out of range.

In our case, for proof of concept, we used an LED which was tied to the water temperature being too high. We set the water temperature range to 30°C and the LED to flash for ten seconds simulating that the action was being done. Once the ten seconds were up, the system would check again to see if the temperature had gone down. If the temperature went down, the script would not do anything and continue collecting data and storing it on the spreadsheet. If the temperature was still too high, the system would continue to collect data, but the LED would run for another ten seconds.

As you can see in Figure 2 above, we run into our first complications and one of the limitations of the Google API solution. In the case the JSON file isn't properly configured correctly, the raspberry pi becomes stuck and cannot proceed with the collection of data. A problem with the main script is that if a sensor is not working properly, the script will infinitely wait for the sensor to respond before it sends any data.

Google API Side

As the script continues to run, Google API will periodically monitor the last appended line that was added to the spreadsheet. If the last appended value was out of a certain range, the spreadsheet has a script that will automatically send a notification to the user.

The script (See Appendix B: Code Essentials is enabled by starting the function. The function periodically checks the last line of the spreadsheet to see if the last value entered was above a certain range. For our use, the raspberry pi was collecting data at a rate of 1 data entry/second. Since temperature does not change drastically in a matter of seconds, we opted to have the Google spreadsheet function check every minute. This value can be changed depending on the user's preference.

In the case that the function detects a value which is deemed *'out of range'* it would take the time, the date, and the temperature of the value which was out of range. The notification would be sent to the email of the owner.

A flaw that comes with this function is that if, the temperature does not lower within the time of the next check, the function will continuously send the owner notifications about the value being out of range.

As we mentioned in the previous section, the Google API solution is very limited. Although we were able to store the data, there is no clear way to fix the endless stream of notifications if the problem is not solved by the automatic actions done by the system. There is also the problem of the spreadsheet becoming too large to check daily since it would have a thousand plus rows being appended every day.

Schematic

A full analysis of the schematic design can be found in Appendix C: Schematic Diagram but the design of the schematic was based on a couple of guides (which are also mentioned in the appendix). We made use of a solderable breadboard (Figure 4) which allowed us to take the components off a regular breadboard and place them onto a permeant breadboard for casing purposes.

Figure 4: Solderable Breadboard

We used jumper female/male jumper cables to connect the breadboard to the raspberry pi, and we used short shielded copper wires to make connections on the breadboard.

Results

Our project was a huge success, we were able to complete the project on time with all the design specifications we had set at the beginning of development. A sample of how the data collected would look like can be seen in Table 1. For reference, all testing was done with the raspberry pi collecting data once every second, and the notifications script checking every minute.

Table 1: Sample of the spreadsheet data

Time and Date	Room Temperature C	Water Temperature C	Room Humidity %
14:13 11/14/18	23	23.187	45
14:13 11/14/18	22	23.187	46
14:13 11/14/18	23	23.187	46
14:14 11/14/18	23	23.187	46
14:14 11/14/18	23	23.187	46
14:14 11/14/18	23	23.25	46
14:14 11/14/18	23	23.187	46
14:14 11/14/18	23	23.187	46
14:14 11/14/18	23	23.187	46
14:14 11/14/18	23	23.25	45

As you can see, the data being collected was taken at a collection rate of 1 data entry per second. We were not able to do a functional test over a longer period since we were running off a hotspot. The testing period over several hours also required us to stay with the device and for the hotspot to be on constantly. We also could not work on the device if we had tested it over a longer period. The idea, however, is there and it shows that data can be collected and pasted onto a spreadsheet on Google Drive.

Figure 5 shows the graph that could be made from the data that was collected from the sample table above.

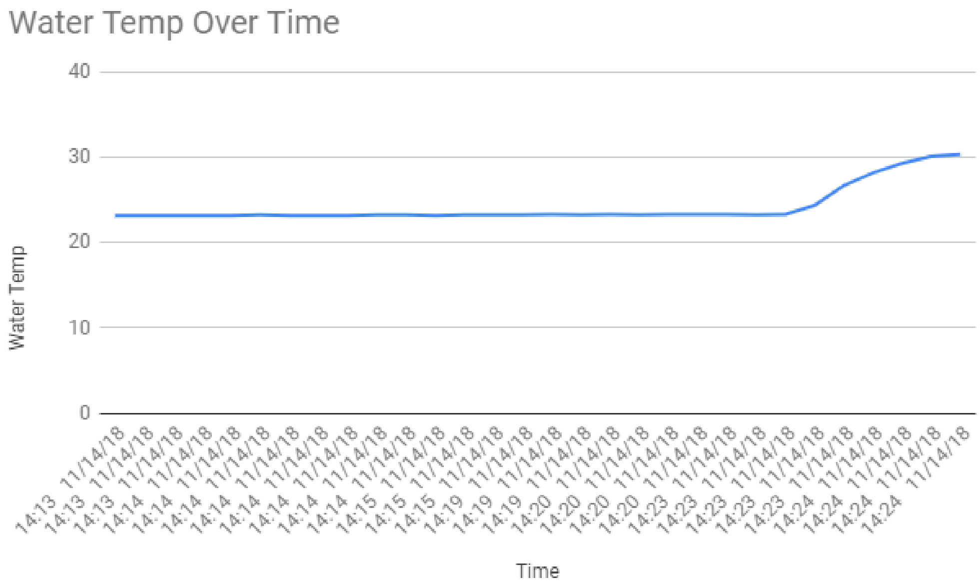


Figure 5: Graph showing trend of temperature over time

As you can see from the graph, the temperature over time can be shown on the spreadsheet; however, there is a problem creating this figure. Google spreadsheets did not cooperate with how the raspberry pi was appending new rows to the spreadsheet. This graph above was created after a testing period which isn't how we wanted to display the data. We were able to find a way to graph the data while simultaneously appending data to rows, but it wasn't a solid solution since it slowed down the spreadsheet operation.

Ultimately after many complications with the spreadsheet operation, we decided to not graph the data synchronously with the collection of data for the sake of keeping the CPU usage for the spreadsheet low.

Notifications and actions worked the way we had designed them to work, but there were some flaws with the operation. The actions worked, but there was a delay between the temperature being too high and the action starting. We couldn't time the delay, but from we could see there was a very slight delay between the LED lighting up for 10 seconds and the temperature reaching a level above 30°C. We concluded that the delay was due to the processing speed of the raspberry pi. The raspberry pi was doing multiple things at once which might have slowed down the speed at which the script would compare the recently pulled data to the set point.

The notifications also worked similarly to the actions in the sense that there would be a delay between the comparing of the data and the set point. Since the script was programmed to check every minute, there would be a delay. Due to the periodic checks, the reported temperatures may not be exact. In theory, the actions would have already started attempting to reduce the temperature of the system. If both are not in sync, the actual temperature and the reported temperature may not be correct.

Complications

We ran into many problems with the use of Google API there were also some other minor problems with the sensors. However, the main source of problems was the use of Google API to which we have very little solutions to. Since we had a tight schedule at the time, we didn't have many choices to choose from, and in hindsight it might have been better to take that extra time to create our own site. But once again, due to the tight schedule and a consensus amongst the team to see how it will operate we continued with the Google API.

Google API

Our main problems with the API were:

- User-friendly interface
- Correct credentials for the JSON file
- Script appending rows to the spreadsheet
- Ability to switch spreadsheets
- Notification

The need of creating a JSON file with the credentials of the Google account with the spreadsheet isn't very 'user-friendly.' In the case that someone decides to use what we have; the individual would have to know how to code to adjust any of the files. This isn't how we wanted to direct our project initially, but we decided to base this project not on how the product would do on the market, but rather for an individual who is a hobbyist.

With the script continuously appending rows to the spreadsheet, it would begin to slow down. The speed was slowed down enough that the raspberry pi would be appending rows on the console screen but wouldn't be appearing on the spreadsheet until a second or two after. But by then there would have already been two more data points that the raspberry pi would have already collected. A single Google spreadsheet would be able to store enough data for about eleven months to a year's worth of data if the data was sampled every second.

We had wanted to switch the spreadsheets out after a week or a month of use. But we could not do anything of that sort with Google Docs. With the desire to also graph all the data in one spreadsheet, a year's worth of data would take a significant amount of time to process assuming the sample rate is once every second. This problem is easily avoided by changing the sample time.

Notifications had several problems when we were testing. After a certain point, the notifications would not work after the initial start-up of the function. We knew that the function would check every minute, but there were times where the function would be running, and the temperature would be high for longer than a minute without notification. This is a minor flaw since the autonomous action would be running, but there wouldn't be any notification that would notify the owner of the temperature rise which could be a problem on the actual system.

Another problem which occurred with the notifications was the number of notifications being sent if there was a problem. During testing, if there was a problem for a long period of time, it would constantly send notifications to one email account if the script was running. In the case that the collection of data stopped, and the last row was above 30°C, the notifications would continue to send even if the data was not being collected. This problem occurs since both scripts are running independently, and since the Google spreadsheet script is based off the last appended row. The script will continue to check every minute and send a notification unless you turn on the system again and have it appended a value lower than 30°C.

Other Complications

At the beginning of development, we ran into a problem with parts. Originally, we had planned to order parts from eBay and Amazon. Both options would cut down on costs of the project the only problem with cutting the cost down was the time of arrival for the parts. The estimated delivery time from eBay was either mid-October or the end of November. Both were not valid options since the project was to be completed before the end of November and we could not count on the eBay delivering on-time. BCIT did not have Amazon Prime, and we could not guarantee the shipment date.

We encountered problems with the sensors. While using a guide for both sensors, we noticed that both guides used the same GPIO port on the raspberry pi. This was easily resolved when we looked up the raspberry pi data sheet (See Appendix E: Raspberry Pi Data Sheet and found there were multiple GPIO ports.

Another sensor problem we witnessed was with the humidity/temperature sensor. Occasionally it would suddenly take in a reading which was way out of range. It would suddenly jump from 23°C and 46% humidity (for example) to 11°C and 150% humidity but then suddenly return to normal temperatures in the next second. We concluded that it was a glitch, and it was a problem with the sensor itself and not our system. The other sensor did not experience the same glitch and so we concluded that it was a problem with the sensor itself.

A very large problem, which we found out at the end of testing, was the auto-start. Our entire system is reliant on the Wi-Fi connection to the raspberry pi. This means that in the case that the raspberry pi was to auto-start with no Wi-Fi connection, the system would not auto-start and would sit there idle. The only way to resolve this problem was to ensure our testing hotspot was on and to restart the system manually.

Cost

All Parts in Table 3 of Appendix D: Bill of Materials was what the team original was going to use to develop the aquaponic data collection system. The budget for the project was $200, but one of our primary goals was to create a system at the least possible cost. So, we picked parts that would be enough to demonstrate the system without going overboard.

We went with a raspberry pi since an Arduino wouldn't be enough to handle all the processes that we wanted our main module for the system. We also wanted to send the data over Wi-Fi, for those reasons we decided to go with the raspberry pi 3. The raspberry pi is basically a smaller computer compared to an Arduino which is more of a microcontroller, and the raspberry pi 3 model B comes with a built-in Wi-Fi adapter.

We also had planned to make use of three sensors that would monitor the system. Altogether the system would come to a cost of $136.88, at the time this was considered the lowest cost we could get the project to be. However, after receiving the bell siphon solution (Appendix A: Bell Siphon, we cut most of the cost and the actual parts list for the project can be seen in Appendix D: Bill of Materials

With the changes, we save a total of $22.11. Although we were almost under $100 to push the envelope further to fulfill the *"as low as possible"* focus, we decided to see if we could source as many items as possible for free. So, we were able to source the raspberry pi from a colleague who already graduated, an SD card from a stakeholder, and a chassis from an old storage room from the school.

Error! Not a valid bookmark self-reference. shows the efforts of our *'extreme'* saving. The overall cost of the project was reduced by a total of 80% due to sourcing many of the materials for free. We could have pushed the price further down if we had supplied the parts of our project from eBay or Amazon. If the parts were bought from either eBay or Amazon, the total cost of both sensors would have been $5-$10.

Table 2: Actual BOM for Capstone Project

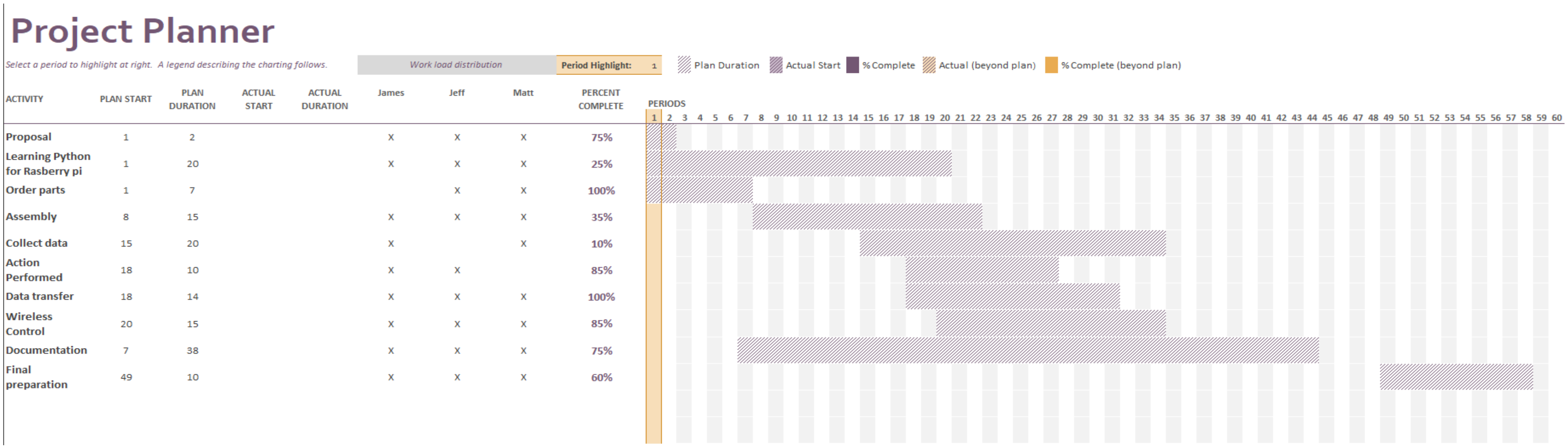
Part Name	Purpose	Quantity	Cost (tax not include)
DS18B20 Thermometer Temperature Sensor Probe Module	Water Temperature Sensor	1	13.25
DHT11 Temperature And Relative Humidity Sensor	Room Temperature and Humidity Sensor	1	8

Total Cost + tax	
$	23.80

However, we wouldn't be able to order from Amazon through the school since BCIT did not have Amazon Prime. The parts from eBay had an estimated time of arriving from either the middle of October to the end of November. Both times were not possible because the development of the project would have been delayed waiting for the sensor to arrive. If we had gone with eBay, there was without a doubt in the team's mind that we wouldn't finish in the given time.

Schedule

Figure 6: Gantt Chart for our Project displays our initial Gantt chart for our project. We had planned to finish one week before the presentation date. We also planned to create a website that would be displaying all our data. But since that was cut from the project and a different solution was used instead the schedule changed.



Project Planner

Select a period to highlight at right. A legend describing the charting follows.

Work load distribution | Period Highlight: 1 | Plan Duration | Actual Start | % Complete | Actual (beyond plan) | % Complete (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	James	Jeff	Matt	PERCENT COMPLETE
Proposal	1	2			X	X	X	75%
Learning Python for Rasberry pi	1	20			X	X	X	25%
Order parts	1	7				X	X	100%
Assembly	8	15			X	X	X	35%
Collect data	15	20			X		X	10%
Action Performed	18	10			X	X		85%
Data transfer	18	14			X	X	X	100%
Wireless Control	20	15			X	X	X	85%
Documentation	7	38			X	X	X	75%
Final preparation	49	10			X	X	X	60%

Figure 6: Gantt Chart for our Project

Figure 7 roughly shows the updated schedule we followed. The differences between the one above and the revised one are that the amount of time to set up the Google account didn't take as long as we had planned the website to take. We were always testing the sensors every work session, and the time it took to graph and do actions only took one day.

The reason for such a large time was to account for the courses we were taking at the same time. The team met every Friday to complete the project, and closer to the due date we started attending Wednesday morning and afternoon to the work sessions.

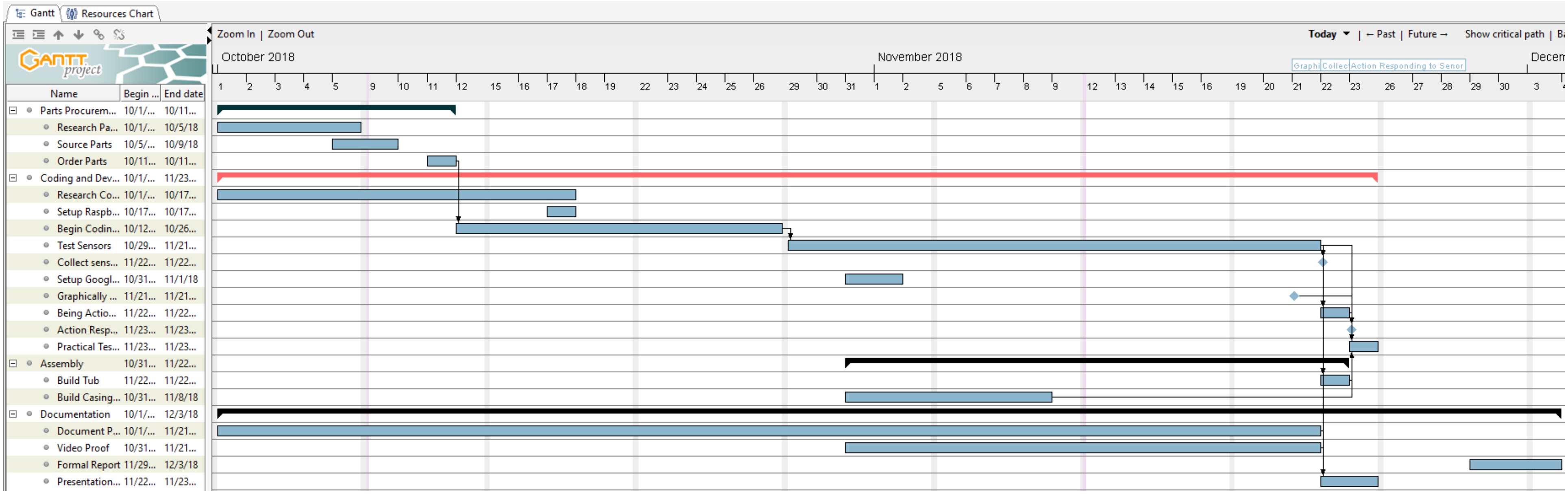


Figure 7: Revised Schedule

Conclusions

The automated data collection system we were able to prototype for the SFU student can execute what he wants out of it with flaws. The system can collect data from two sensors and append the collected data to a web server (Google Drive). The system can autonomously active actions based on readings from the sensors (LED). The spreadsheet monitors incoming data and sends a notification to the owner of a possible problem with the system (Email notification).

Google Drive can be accessed from any device at any time. However, it becomes a problem when the spreadsheet, the system is appending data to, reaches a certain size. The file may become too large and unopenable if it reaches a certain size, but this is a theory and it was not tested since we could not test it for that long. The flaws of the system may hinder its ability to be expandable for the SFU student. However, the SFU student is well versed in programming and can probably pick up the slack and find better solutions to the flaws.

Recommendations

1. Improve the script functionality
   a. As of right now, the script will run if all conditions are met, but it will become *'stuck'* if one condition is not met.
   b. The script will also become *'stuck'* if one of the sensors does not work. It will wait till the sensor gets a reading before sending the data.
   c. If a sensor does stop working the script will append a "Sensor Broken" line instead of data to the spreadsheet
   d. The script has no way of creating a new spreadsheet to enter data in, if the name of the script spreadsheet is not on the Google Drive it won't start

2. Create a way for the data to be collected even if there is no network connection

3. Find another solution for storing the data in a more reliable and secure way.

4. Create a way to auto-restart the raspberry pi in the case of it not collecting data.

5. Source as many parts for free to keep the cost of the system low.

6. Add more sensors to the system

Citations

[1] “What is Aquaponics,” *The Aquaponic Source*. [Online]. Available: https://www.theaquaponicsource.com/what-is-aquaponics/. [Accessed: 27-Sep-2018].

[2] “Turning on an LED with your Raspberry Pi's GPIO Pins,” *The Pi Hut*. [Online]. Available: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins?fbclid=IwAR3dOaGQPdjA3zsCdL524iK9IRESBE_H3k8_KwZL66AaR7MsmG_f7ab7_oo. [Accessed: 02-Dec-2018].

[4] “Ibc Aquaponics Chop System,” *Home Security RSS*. [Online]. Available: https://homesecurity.press/quotes/ibc-aquaponics-chop-system.html. [Accessed: 03-Dec-2018].

[5] “Setting Up Authentication for Server to Server Production Applications | Authentication | Google Cloud,” *Google*. [Online]. Available: https://cloud.google.com/docs/authentication/production?fbclid=IwAR03wpJmTKxhf1GVCVKfAW77hC1AQpdXnxer_5nxPBBgFUv3yezIuC_5juU. [Accessed: 03-Dec-2018].

[6] “Setup a Raspberry Pi Temperature Sensor,” *Pi My Life Up*, 25-Oct-2018. [Online]. Available: https://pimylifeup.com/raspberry-pi-temperature-sensor/. [Accessed: 03-Dec-2018].

[7] “Raspberry Pi: Measure Humidity and Temperature with DHT11/DHT22,” *Raspberry Pi Tutorials*. [Online]. Available: https://tutorials-raspberrypi.com/raspberry-pi-measure-humidity-temperature-dht11-dht22/. [Accessed: 03-Dec-2018].

[8] “Particle,” *Particle Datasheets Documentation | Photon datasheet*. [Online]. Available: https://docs.particle.io/datasheets/discontinued/raspberrypi-datasheet/. [Accessed: 03-Dec-2018].

[9] “TEMPERATURE SENSOR - WATERPROOF (DS18B20),” *Lee's Electronic Components*. [Online]. Available: https://leeselectronic.com/en/product/1255.html. [Accessed: 03-Dec-2018].

[10] “HUMIDITY AND TEMPERATURE SENSOR DHT11,” *Lee's Electronic Components*. [Online]. Available: https://leeselectronic.com/en/product/7374.html. [Accessed: 03-Dec-2018].

Appendix A: Bell Siphon

A bell siphon is a way of draining excess water from the growth bed of an aquaponic garden. It utilized the physics behind fluid dynamics which was a course taken by the SFU student. He found the design on the Internet and told us that we wouldn't need a level sensor in our prototype. Figure 8 shows an example of how a bell siphon is set up in an aquaponic system.

Source: valschep, "Best bell siphon ever explained," *YouTube*, 17-May-2013. [Online]. Available: https://www.youtube.com/watch?v=lyrvcCqv5V0. [Accessed: 03-Dec-2018].

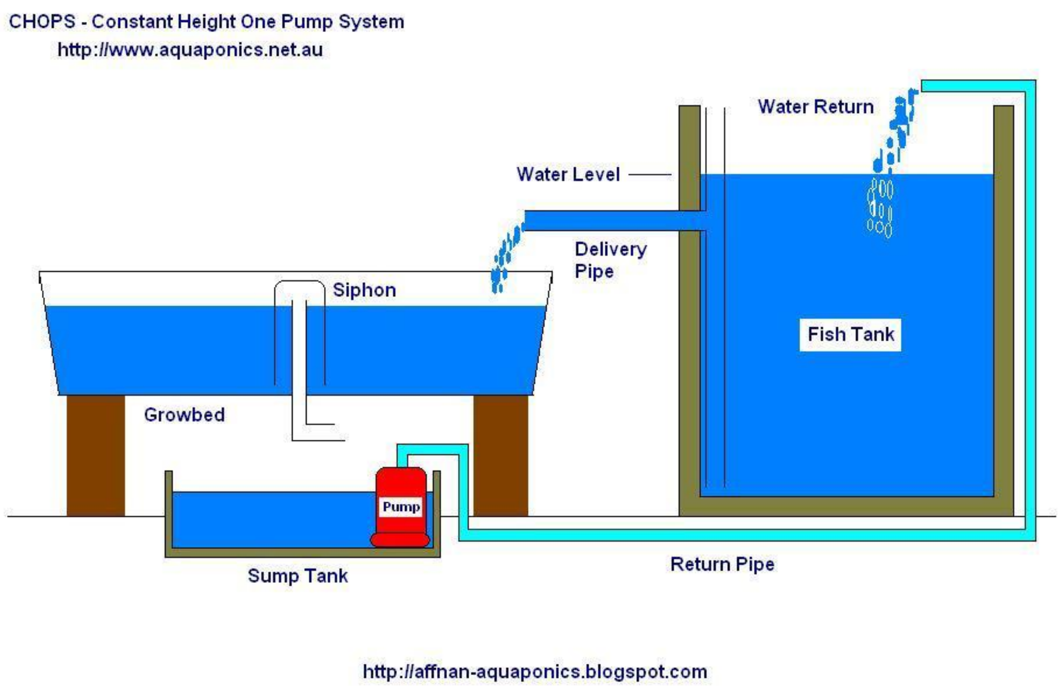


Figure 8: Bell Siphon example

The process is split up into four phases; each phase refers to the level of the siphon. The first phase is a filling phase, the level of the siphon begins at the bottom where there is a hole under the 'bell.' Here, water enters from the growth bed and will fill up to the top of the bell. Once the water level reaches the top of the bell, it reaches phase two.

Phase two is a water lock, once the water reached the top, the water will begin to drain into the middle pipe. This creates an air pocket in the middle pipe which will slowly become smaller as more water pours in.

Phase three is when the water reaches full capacity in the bell and the air pocket is gone due to the excess water flowing in, creates a vortex which sucks out all the water in the growth bed and pulling it to the middle pipe. This phase is called the draining phase since all the excess water is being pumped out of the growth bed and into the sump tank

Once the water level reaches the bottom of the open portion of the bell again, the siphon will begin to break which will allow air back into the siphon. This phase is called *'Breaking Siphon.'* Once air has filled the siphon and all the water has drained out of the main pipe the cycle begins again.

This cycle eliminates the need for a water level sensor since the bell siphon will continuously drain out water if the level in the growth bed becomes too high. The water in the fish tank may need to be refilled, but it uses the overfilling of the tank to send water into the system which also eliminates the need for a water level sensor.

Appendix B: Code Essentials

Appendix B.1: Auto-start script

The code below are two separate scripts which was first used to open the console (after the raspberry pi has powered up), then enter the right folder to run the second script. The second script opens the actual script which will open the main script that runs the operation of the sensors.

Script "Start" was created in: /home/pi/start

The "Start" script contains and executes the following commands:

cd /home/pi/Adafruit_Python_DHT/examples

./Aqua.py

To auto-start the *'Start'* script, we would need to add a line to the *'autostart'* file located here: *home/pi/ .config/lxsession/lxde-pi/autostart.*

The following line was added *(This command points to the "start" script created in the previous step):*

@lxterminal -e bash /home/pi/start

This enable the raspberry pi to run the *'Aqua.py,'* script from its location. (This is due to the dependencies of the other file)

Appendix B.2: JSON File Authentication

The JSON file was created using the Google API interface. This JSON file will give permission to the raspberry pi to append rows and input the data from the sensors onto the spreadsheet.

The guide we used can be found here: https://cloud.google.com/docs/authentication/production?fbclid=IwAR03wpJmTKxhf1GVCVKfAW77hC1AQpdXnxer_5nxPBBgFUv3yezIuC_5juU

The "private key" and ID for each section is for testing purposes. If the guide is followed properly, the JSON file with other credentials will output a similar code.

Code:

"type": "service_account",

"project_id": "aquaponic-system-data",

"private_key_id": "a4787b4eae8d1b528ad487868da3a13e79313544",

*"private_key": "-----BEGIN PRIVATE KEY-----
\nMIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDSD811DbCdKX/S\nc/H
Ksji8b1YMIaxga9xRnPKh1g8R4crUvQUdMkau7gVjSFkOilvM88PDcVjXT07A\nEAdhzN3+tSED
qVQWHSXH1/jhAdh1NVsE8LXgDb++sfFslU4TK3k0iu6V+W/RRpCT\nED2Tm8dU6WMlWUziV
kkx6WKEg0xfWVgD3ysmvLPmicJcF3PbOKJuvrsRfXaBKE98\nXEBcvOdVX8TcJtp3Utz8nJeixS0
q68Eql3ovU9j6uKgeXaTVruhyROD9PER/VIOp\nM/GCcDDERmHyccGxrQq1olzi9pq9TN2c03L
UFrgkIzVTR6fmGxDTBGQ0fA+sLiLd\n/ERrqm9PAgMBAAECggEAQ3tDIBbhRWPHBdcsdlbRpi
tII8aPhzwvP2KhiX7oQXDd\nRYFhirMThO+++FmODn9BJXKr0XXwud5m6IldrbvwFvqeG/yCIE
aF1BDcCYCbxm95\nCXnAm7UfA4ZLjfva5IqjOc9l/PTwSHUYTb46Ydh57Ru6t41cBxv8EJYuiW5
akwhd\n3vikvdO8IG5XCDUfdtgZazRb2xxVGSMuukEJ6goObJNlMXcPT2TXdJkHx/9d+xOQ\nbb
S1uelP3J7QV2JrCDJSBQEz6lqwZs2XHP44Q/g/znRe2zyQl8ugwzUAf5dVo5Ed\nsP8uiK8Gr+6L
wO7gcp/w7aUaKeNHcsP5tZpyEouT0QKBgQD19qaPfbx/AYp/g7pB\nsCDtG1udxH4vH/gGrCT
cVlV6s6Hn1LY2WvZ9zB0VAYb+UYv+fNiV2ul9kTUV7k/U\n26DDISTpVNV2cJEczWDmtarRGn
ehS2pNfL4O/vI+2p3CI0mn8G5VK1kK8aQHd5pJ\neyNQa9iwUXHzfXQL68sTNayskQKBgQDao
h6+AUj+csb7OBTshOVUC8nl4M5GBKU0\nqnOpzrOiavDkvvdgin/tk/HY9kKjxPtm6F1sFT8T/o
IbNe+dhQDujXnQnnpos8eq\nKqTvsa3wYInVth65JALKVf4OpMTU8VtxIcpX4vQrwPqdMNQfYgt
HcgKydVAYmE18\nxMyCaxDN3wKBgQC83k0/2fkoPXKwP+KCFlDyQRO0rfGHFjFKTl1vpAu3U
MB+WGQi\nkO694kJiou2Z09x/6jYVvCeW+C2fTYOXm55yt7y0PyQ68m07hKln/w/PyVcbT+Gs\
n5qC6uyIhCZYiQR6KCqFuXP7e3HZsFXcv/ZI8atPVcHTj1+2ISeOCOLEDQQKBgQDO\nTMvVXkE
SWBuKUZDs3TnH5JfOYiJBz4BUAcLCKFk4mNBWCBxE7GtqMn+yyix3fBf9\nMqV0p1et9imuo2
ZGrIoEj1qyj9fRlFKDL1oTEwWeHHnzPrYe3OwFIL9p8fAJPvWZ\n+tiDtZPQ9XeFe92s/0bFU79X
xXqeupfjc4/GQGDK4QKBgQDhuOUOQeOy1XsM7bIY\npGop5ItUjcXcrq+Q2TlcnaYZUx3TOrn9Y
TPl/kx0cB2gxrerJH/h+sFz+uq6Dgcn\nH3MUT8rRHWc3QlNjjc58xwwFi/OjRtZNhwG3xjFJSoS
bS8Gx6eXdM9/QuqvvrqGL\nJjcS35Wr/8LZY0GWtPSk5Q5Qeg==\n-----END PRIVATE KEY-----
\n",*

"client_email": "test9000@aquaponic-system-data.iam.gserviceaccount.com",

"client_id": "100055336935501261729",

"auth_uri": "https://accounts.google.com/o/oauth2/auth",

"token_uri": "https://oauth2.googleapis.com/token",

"auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",

*"client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/test9000%40aquaponic-system-*

data.iam.gserviceaccount.com"

Appendix B.3: Main Script Code

The main script of our code is responsible for the operation of the sensor collection of the data. Most of the code is open source which can be found on the separate sites below.

DS18B20 Water Temperature Sensor: https://pimylifeup.com/raspberry-pi-temperature-sensor/

DHT11 Room Temperature and Humidity Sensor: https://tutorials-raspberrypi.com/raspberry-pi-measure-humidity-temperature-dht11-dht22/

“Actions”: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins?fbclid=IwAR3dOaGQPdjA3zsCdL524iK9IRESBE_H3k8_KwZL66AaR7MsmG_f7ab7_oo

The code below is a combination of both guides using mainly the open source given by the publisher of the DHT11 sensor. The script below is responsible for the operation of both sensors, the appending of data collected to the spreadsheet, and the actions associated with the temperature readings.

Code:

```
#!/usr/bin/python

# Google Spreadsheet DHT Sensor Data-logging Example

# Depends on the 'gspread' and 'oauth2client' package being installed.  If you
# have pip installed execute:
#   sudo pip install gspread oauth2client

# Also it's _very important_ on the Raspberry Pi to install the python-openssl
# package because the version of Python is a bit old and can fail with Google's
# new OAuth2 based authentication.  Run the following command to install the
# the package:
```

```
#   sudo apt-get update
#   sudo apt-get install python-openssl

# Copyright (c) 2014 Adafruit Industries
# Author: Tony DiCola

# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
import json
import sys
```

```
import os

import glob

import time

import datetime



import Adafruit_DHT

import gspread

from oauth2client.service_account import ServiceAccountCredentials


#code added for actions

import RPi.GPIO as GPIO



# Type of sensor, can be Adafruit_DHT.DHT11, Adafruit_DHT.DHT22, or
Adafruit_DHT.AM2302.

DHT_TYPE = Adafruit_DHT.DHT11


# Example of sensor connected to Raspberry Pi pin 23

DHT_PIN  = 5

# Example of sensor connected to Beaglebone Black pin P8_11

#DHT_PIN  = 'P8_11'


# Google Docs OAuth credential JSON file.  Note that the process for authenticating

# with Google docs has changed as of ~April 2015.  You _must_ use OAuth2 to log

# in and authenticate with the gspread library.  Unfortunately this process is much

# more complicated than the old process.  You _must_ carefully follow the steps on
```

```
# this page to create a new OAuth service in your Google developer console:
#   http://gspread.readthedocs.org/en/latest/oauth2.html
#
# Once you've followed the steps above you should have downloaded a .json file with
# your OAuth2 credentials.  This file has a name like SpreadsheetData-<gibberish>.json.
# Place that file in the same directory as this python script.
#
# Now one last _very important_ step before updating the spreadsheet will work.
# Go to your spreadsheet in Google Spreadsheet and share it to the email address
# inside the 'client_email' setting in the SpreadsheetData-*.json file.  For example
# if the client_email setting inside the .json file has an email address like:
#   149345334675-md0qff5f0kib41meu20f7d1habos3qcu@developer.gserviceaccount.com
# Then use the File -> Share... command in the spreadsheet to share it with read
# and write acess to the email address above.  If you don't do this step then the
# updates to the sheet will fail!
GDOCS_OAUTH_JSON       = 'Aqua.json'

# Google Docs spreadsheet name.

#code for diffrent sheets:
#sheet = raw_input("What's the name of the spreadsheet?  ")
#type(sheet)

sheet = 'test'

GDOCS_SPREADSHEET_NAME = sheet
```

```
# How long to wait (in seconds) between measurements.
FREQUENCY_SECONDS      = 1




def login_open_sheet(oauth_key_file, spreadsheet):
    """Connect to Google Docs spreadsheet and return the first worksheet."""
    try:
        scope =
['https://spreadsheets.google.com/feeds','https://www.googleapis.com/auth/drive']
        credentials = ServiceAccountCredentials.from_json_keyfile_name(oauth_key_file, scope)
        gc = gspread.authorize(credentials)
        worksheet = gc.open(spreadsheet).sheet1
        return worksheet
    except Exception as ex:
        #print('Unable to login and get spreadsheet.  Check OAuth credentials, spreadsheet name,
and make sure spreadsheet is shared to the client_email address in the OAuth .json file!')
        print('Google sheet login failed with error:', ex)
        sys.exit(1)



print('Logging sensor measurements to {0} every {1}
seconds.'.format(GDOCS_SPREADSHEET_NAME, FREQUENCY_SECONDS))
print('Press Ctrl-C to quit.')
worksheet = None
while True:
```

```
# Login if necessary.
if worksheet is None:
    worksheet = login_open_sheet(GDOCS_OAUTH_JSON, GDOCS_SPREADSHEET_NAME)

# Attempt to get sensor reading.
humidity, temp = Adafruit_DHT.read(DHT_TYPE, DHT_PIN)



#water temperature sensor
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
```

```
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c

# Average out data over 5 min period of time

#temp_avg =
#humidity_avg =
#room_temp_avg =

#Time and Date

timenow = datetime.datetime.now()
time_string = timenow.strftime("%H:%M   %x")
time.sleep(2)

#code for actions
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
if(read_temp() > 30):
   GPIO.output(18,GPIO.HIGH)
   time.sleep(10)
   GPIO.output(18,GPIO.LOW)
else:
```

```
    GPIO.output(18,GPIO.LOW)

  # Skip to the next reading if a valid measurement couldn't be taken.
  # This might happen if the CPU is under a lot of load and the sensor
  # can't be reliably read (timing is critical to read the sensor).
  if humidity is None or temp is None:
    time.sleep(2)
    continue

  print('************************************************')
  print('Current Time and Date:   '+ time_string)
  print('Room Temperature: {0:0.1f} C'.format(temp))
  print('Water Temperature: {0:0.1f} C'.format(read_temp()))
  print('Room Humidity:    {0:0.1f} %'.format(humidity))
  print('************************************************')

  # Append the data in the spreadsheet, including a timestamp
  try:
    worksheet.append_row((time_string, temp, read_temp(), humidity))
  except:
    # Error appending data, most likely because credentials are stale.
    # Null out the worksheet so a login is performed at the top of the loop.
    print('Append error, logging in again')
    worksheet = None
    time.sleep(FREQUENCY_SECONDS)
    continue
```

```
  # Wait 30 seconds before continuing
  print('Wrote a row to {0}'.format(GDOCS_SPREADSHEET_NAME))
  time.sleep(FREQUENCY_SECONDS)
```

Appendix B.4: Google Spreadsheet Notifications

The script for enabling Google spreadsheets to send notifications to an email address was created using the help of another guide. This script is responsible for detecting if the last line of the spreadsheet is over a certain value. The timing for the notification is done in another window. There was no source code for this script. However, we followed the syntax given by the guide down below.

Guide: https://mailparser.io/blog/send-emails-from-google-spreadsheet/?fbclid=IwAR20-g-28Cs2jj03Pq96youxvRWIdblAwjdhU3un-S9sSjzQu1RU_Oer1wE

Code:

```
function myfunction()
{

 var sheet = SpreadsheetApp.getActiveSheet();
 var last_Row =  SpreadsheetApp.getActiveSheet().getMaxRows(); // First row of data to process
 var numRows = 1; // Number of rows to process
 // Fetch the range of cells A2:B3
 var dataRange = sheet.getRange(last_Row, 1, numRows, 5);
 // Fetch values for each row in the Range.
 var data = dataRange.getValues();
 for (i in data)
 {
  var row = data[i];
```

```
  var date = row[0]; // First column
  var room_temp = row[1]; // Second column
  var water_temp = row[2]; //third column
  var humidity = row[3]; //fourth column

 }
 if (water_temp > 30)
 {
  var subject = 'Aquaponic System WARNING - Water Temp High!';
 var message = date  + "\n" + ' water temperature is ' + water_temp;
  MailApp.sendEmail('sampleemail@gmail.com', subject, message );

 }

}
```

Appendix C: Schematic Diagram

Figure 9 below shows the schematic diagram that we used to connect the sensors to the raspberry pi. The guides previously mentioned in Appendix B.3: Main Script Code gave us insight for setup the sensors.

The DS18B20 (water temperature) sensor needed to have wires soldered onto the pins since we could not directly connect the leads onto a breadboard. The leads of the DS18B20 were twisted pair and if we directly connected them onto a breadboard the wires could become frayed which would damage the lead.

For the DHT11, the leads were very short and to prevent damaging the actual sensor itself, we mounted the sensor onto a 16-pin mounting base and cut-off the other 4 pins.

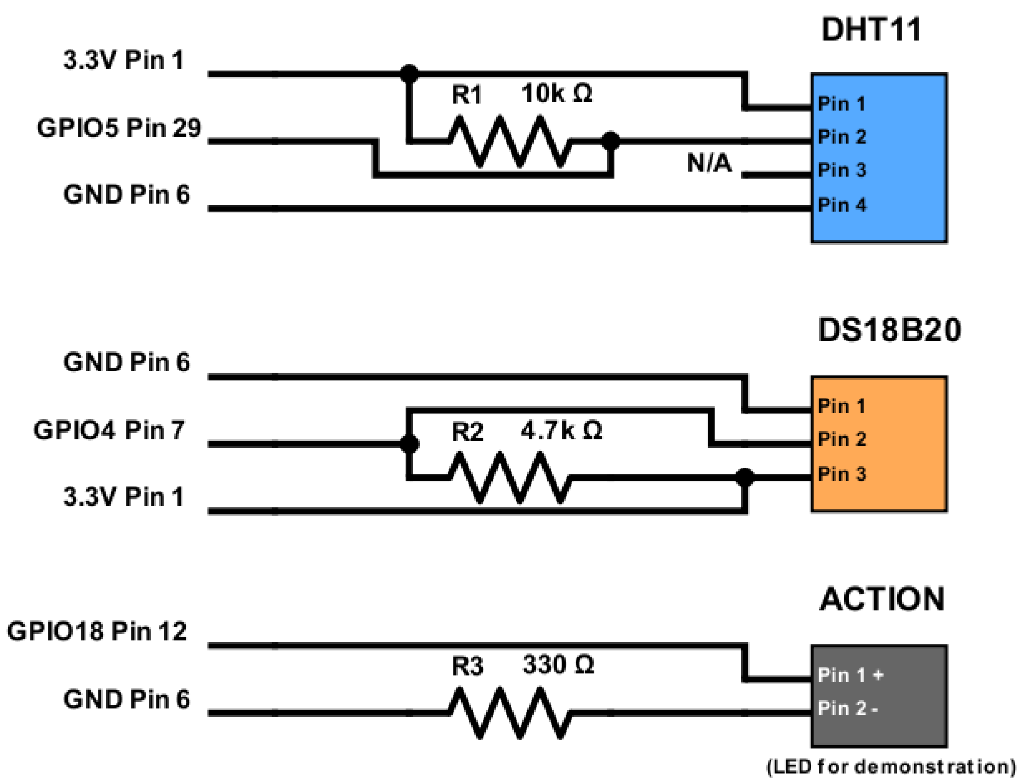


Figure 9: Schematic Diagram for sensors to Raspberry Pi

Appendix D: Bill of Materials

The original bill of materials (BOM) for our project can be seen below in Table 4: BOM after design modifications The first table shows the initial BOM with all the parts that we thought we were going to implement into our system. After some design changes, we decided to eliminate the water level sensor and water pump. These two would be replaced with a bell siphon (Appendix A: Bell Siphon which uses a much more natural solution to maintain the water level.

Table 3: Initial BOM for capstone project

Part Name	Purpose	Quantity	Cost (tax and shipping not included)
Raspberry Pi 3 Model B	Main processing module for our system	1	$55.99
Dc 3.5v -9v 3w Usb Submersible Water Pump	Water tank Level Adjustment	1	$17.99
DS18B20 Thermometer Temperature Sensor Probe Module	Water Temperature Sensor	3	$14.68
DHT11 Temperature And Relative Humidity Sensor	Room Temperature and Humidity Sensor	5	$22.99
SanDisk Extreme 32GB microSDHC	Data storage and backup	1	$25.23
Total Cost	$ 136.88		

The table below shows what we used, not including the chassis since we could not put a price on how much a chassis would cost. In the end, a chassis can be made from any box.

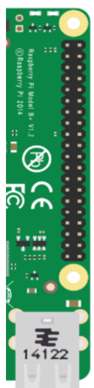
Table 4: BOM after design modifications

Part Name	Purpose	Quantity	Cost (tax and shipping not included)
Raspberry Pi 3 Model B	Main processing module for our system	1	$55.99
San Disk Extreme 32GB micro SDHC	Data storage and backup	1	$25.23
DS18B20 Thermometer Temperature Sensor Probe Module	Water Temperature Sensor	1	$13.25
DHT11 Temperature And Relative Humidity Sensor	Room Temperature and Humidity Sensor	1	$8.00
Adafruit Perma-proto 1/2 sized breadboard	breadboard to be put into the chassis	1	$6.50
Total Cost (with taxes)	$121.92		

Appendix E: Raspberry Pi Data Sheet

The datasheet was found at:
https://docs.particle.io/datasheets/discontinued/raspberrypi-datasheet/

We used this datasheet to find the GPIO of the raspberry pi in order to relocate the pins for our sensors since their guides use the same GPIO port. (Figure 10)



Peripherals	GPIO	Particle	Pin #			Pin #	Particle	GPIO	Peripherals
3.3V			1	X	X	2	5V		
I2C	GPIO2	SDA	3	X	X	4	5V		
	GPIO3	SCL	5	X	X	6	GND		
Digital I/O	GPIO4	D0	7	X	X	8	TX	GPIO14	UART
GND			9	X	X	10	RX	GPIO15	Serial 1
Digital I/O	GPIO17	D1	11	X	X	12	D9/A0	GPIO18	PWM 1
Digital I/O	GPIO27	D2	13	X	X	14	GND		
Digital I/O	GPIO22	D3	15	X	X	16	D10/A1	GPIO23	Digital I/O
3.3V			17	X	X	18	D11/A2	GPIO24	Digital I/O
SPI	GPIO10	MOSI	19	X	X	20	GND		
	GPIO9	MISO	21	X	X	22	D12/A3	GPIO25	Digital I/O
	GPIO11	SCK	23	X	X	24	CE0	GPIO8	SPI (chip enable)
GND			25	X	X	26	CE1	GPIO7	
DO NOT USE	ID_SD	DO NOT USE	27	X	X	28	DO NOT USE	ID_SC	DO NOT USE
Digital I/O	GPIO5	D4	29	X	X	30	GND		
Digital I/O	GPIO6	D5	31	X	X	32	D13/A4	GPIO12	Digital I/O
PWM 2	GPIO13	D6	33	X	X	34	GND		
PWM 2	GPIO19	D7	35	X	X	36	D14/A5	GPIO16	PWM 1
Digital I/O	GPIO26	D8	37	X	X	38	D15/A6	GPIO20	Digital I/O
GND			39	X	X	40	D16/A7	GPIO21	Digital I/O

Figure 10: Pin-out for the Raspberry Pi

Appendix F: DS18B20 Temperature Sensor

The data sheet we used to find out pinout of the water temperature sensor can be found here: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/DS18B20.pdf . The PDF was found on the product page of Lee's Electronics which can be found here: https://leeselectronic.com/en/product/1255.html

Appendix G: DHT11 Temperature sensor

The datasheet we used to find the pinout of the humidity and room temperature sensor can be found here: http://www.circuitbasics.com/wp-content/uploads/2015/11/DHT11-Datasheet.pdf . . The PDF was found on the product page of Lee's Electronics which can be found here: https://leeselectronic.com/en/product/1255.html