The George Washington University
School of Engineering and Applied Science
Department of Electrical and Computer Engineering

ECE 4920W - 30

Final Product Review
Low Cost, Remote Data Acquisition Platform for *In Situ* Biological and Fluidic Measurements

Rachel Zaltz, Computer Engineering
Connor Kraft, Computer Engineering
Phillip Jones, Electrical Engineering

Professor Aslani, Professor Bulusu
April 22, 2020

Abstract (R)

A low-cost data acquisition platform will be designed and assembled to increase the ease of dynamic in situ data collection from confined isolated environments such as incubators. The aim of this project is to record humidity, temperature, PH, turbidity, time-stamped imaging, and various other sensor data at user-specified intervals. The device will remain physically independent of the space outside the incubator by using a modular power supply and network connection. The device will run independently of a power supply for the duration of the experiment and wirelessly send the desired data to a remote user. The device will advance research in cellular and polymetric interactions with indirect measurement and imaging in incubators.

Table of Contents

Table of Figures:

1. Introduction (R)

The low-cost data-acquisition platform will be designed to increase dynamic in situ data collection in confined and isolated environments. Currently, when an experiment is performed within an incubator for a long period of time, lab technicians are unable to open the confined environment until the experiment has been completed in its entirety. This results in a loss of crucial data points as the experiment is progressing within the incubator. Currently, the data cannot be obtained during the duration of the experiment and this results in the loss of many crucial data points. A low-cost data acquisition platform would increase data collection in in situ environments.

Currently, there are only a few products that solve the issue at hand. Some of the best products are from Dickson Data, such as the DicksonOne. These products only allow for a small set of sensors and lack more advanced sensor technology and cameras. They are also very expensive, with the cheapest model being $350. Each individual sensor costs over 100 dollars and requires a paid subscription to use some of the more advanced features. Overall, a product does not exist that meets all of the design requirements for this task.

This project looks to implement a system which can gather data within a closed incubator and send that data to a user on any remote computer. The data is gathered from multiple sensors including temperature, humidity, pH, turbidity, and time stamped images. This data is gathered at a time interval which can be specified by the user to allow for greater flexibility of the system. When data is not being gathered, sensors are turned off in order to save power and conserve energy. This data is then sent to a computer using a Wifi connection so that it may be viewed by the user.

2. Impacts and Effects (P)

2.1. Public Health

Our product has no overall effects or impacts on public health in general. While research generated by the project or by employment of our device may positively impact public health, our system itself does not improve public health.

2.2. Safety

See reasoning for public health.

2.3. Welfare

Our project is also heavily focused on the promotion of ease of access of data acquisition. Having a fully wireless module that performs all functionality via remote GUI, with no impact on environment conditions inside the experiment, is an extremely productive option to the alternatives. This will increase the opportunities to have full experiment length data measurement, as well as make the data acquisition process smoother in general.

2.4. Global

As spoken about in the welfare section, this project will make overall data acquisition easier for the user. In addition, users will be able to run data acquisition experiments in areas without wired power; however, the data would have to be transferred locally at some point instead of being dumped wirelessly. This could have potential to drive this system as an alternative for taking large-scale equipment to remote research sites.

2.5. Social

This project will make taking data from environments like an incubator more convenient and hands free, increasing the social wellbeing of the communities it will be deployed in.

2.6. Cultural

Our project has little to no impact on the cultural standing of the community.

2.7. Environmental

Our project has little to no impact on the environment, as opposed to alternative options. As stated before, our project itself has no impact on this, but the research where it is employed could very well be beneficial.

2.8. Economic

Our project has an economic benefit, in that the cost effectiveness of the product is much more productive than its next best alternatives. Due to the low cost to employ this device, money will be able to be reallocated to other research needs.

3. Overall System Requirements and Specifications (All)

System Level Requirements	**System Level Specifications**
The device will operate with a lithium ion battery pack.	The device shall be able to run on a lithium ion battery pack.
The device will have a long battery life, so the device does not have to be removed from the incubator.	The device shall be able to last on the battery pack for the duration of the experiment (minimum 1 day - maximum 4 weeks).
The device will take pictures of the test environment.	A Pi camera must be used to take pictures of the test environment at user specified intervals.
The device will have a light source for the test environment.	The device must conserve battery by only using the lights when needed (e.g., taking a picture) or when the user specifies.
The device will conserve battery with 2 different modes: sleep and active.	The device shall be in sleep mode when measurements and images are not being taken.
The device will measure pH.	The device shall be able to measure the pH of the solution from 0 to 14 at +/-

	0.001 increments using a pH probe, with an accuracy of +/- 0.002.
The device will measure humidity and temperature.	The device shall be able to measure humidity from 20 to 80% with an accuracy of +/- 5% and temperature from 0 to 50 degrees C with an accuracy of +/- 2 degrees C using Pi HAT sensors.
Th device will measure turbidity.	A turbidity probe shall be used to measure the amount of light scattered by the solids in the solution.
Th device will measure electrical conductivity.	A conductivity probe shall be used to measure the electrical conductivity of the solution from 5 to 200,000 micro-S/cm, with an accuracy of +/- 2%.
The device will be accessible to a remote user.	The device shall send data to a GUI that can be accessed remotely through a desktop and mobile application.
Communication will be enabled via Wifi on the device.	The device should be able to send data to a computer via the built in Wifi module on the Raspberry Pi 3+.

4. Approach to Overall Design (C & R)

4.1. Current Design

The goal of the project is to design a system which can gather data in a closed environment and send this data wirelessly to a remote user. The data gathered is specified by the sensors in section 6.2. The system can do this with only one input from the user, the data collection rate (every 30 seconds), and how long to gather data for (12 hours). These inputs and outputs are outlined in Figure 2 below. The flow of data from the user to system and system to user can be seen in Figure 1 below. Further, Figures 3-5 details the modules in more depth.

4.2. Evolution of Current Design

This design has shifted gears slightly amidst the current pandemic. Due to some constraints, not all testing was completed, and the user interface functions on a very minimal level. The overall goal of the project is maintained; however, we were not able to implement our design in a real testing space due to social distancing measures.

5. System and Subsystems (Modules) Functional Diagrams, Requirement and Specifications (All)

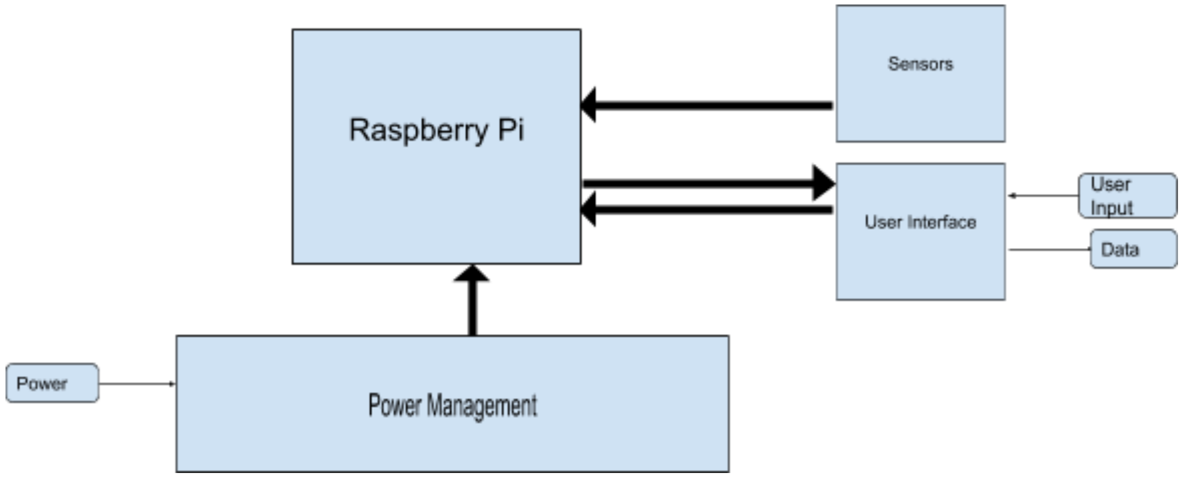


Figure 1- Level 1 Block Diagram

5.1. Raspberry Pi Subsystem Requirements and Specifications

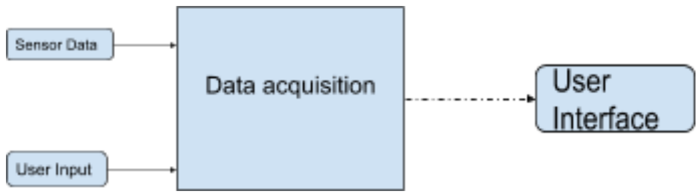


Figure 2- Level 2 Block Diagram Subsystem 1: Raspberry Pi

Raspberry Pi Subsystem Requirements:

- Subsystem provides logic between the data acquisition system and the user interface.

Raspberry Pi Subsystem Specifications

- The Raspberry Pi will take user input and use that to collect data at the specified time.
- The Raspberry Pi will process data collected and send that data to the user interface via Rsync.

5.2. Sensor Subsystem Requirements and Specifications

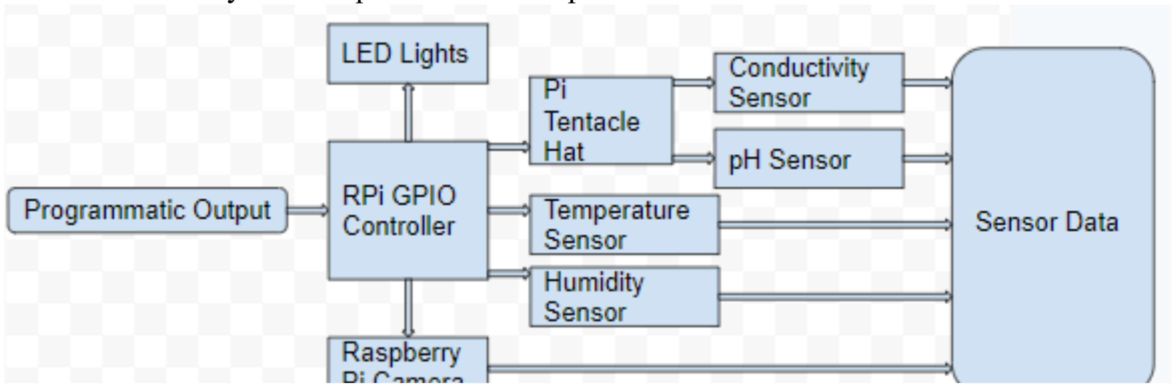


Figure 3 - Level 2 Block Diagram Subsystem 2: Sensors

Sensor Subsystem Requirements:

- Subsystem provides data acquisition for system, using the inputs given via user program instructions.
- Subsystem sends user data back to program data stream.

Sensor Subsystem Specifications:

- The Raspberry Pi GPIO controller will translate the programmatic read instructions to sensors readable inputs.
- The LED will inform each time the system is specified to take a reading.
- The Raspberry Pi camera will take a photo at each read interval.
- The Raspberry Pi Tentacle T3 hat will relay programmatic read instructions to the conductivity and pH sensors.
- The EZO conductivity and pH sensors will gather and relay sample conductivity and pH data at specified intervals.
- The temperature and humidity sensor will process and relay environmental condition data.

5.2.1. Raspberry Pi GPIO Controller Description

The Raspberry Pi GPIO controller provides the basis for sending and reading voltage levels to and from sensors. When provided with corresponding voltage, sensors will return their read value to the header. This information is then reformatted for user or program access.

5.2.2. LED Lights Description

A red LED shall be used to provide a notification system for the user. The light will blink each time a read is made on the system's sensor array.

5.2.3. Raspberry Pi Camera Description

The camera connects to the MIPI camera serial interface on the Raspberry Pi via a 15- pin ribbon cable. It can take 3280 × 2464-pixel images.

5.2.4. Pi Tentacle Hat Description

The hat connects to the GPIO of the Raspberry Pi and outputs 3 BNC connections. It includes chips to handle the configuration of each individual sensor. It passes through the GPIO pins to allow for stacking of hats or connection of other parts.

5.2.5. Conductivity Sensor Description

The Atlas Scientific Conductivity Probe measures system level of conductivity from 5 to 200,000 uS/cm, with an accuracy of +/- 2%. The sensor will relay the environmental or sample level of conductivity at the user specified interval.

5.2.6. pH Sensor Description

The Atlas Scientific pH probe measures system levels of pH from 0 to 14, with an accuracy of +/- 0.002. The sensor will relay the environmental or sample level of pH at the user specified interval.

5.2.7. Temperature & Humidity Sensor Description

The Adafruit DHT11 probe measures environmental temperature and humidity. The humidity sensor measures from 20 to 90 room humidity with an accuracy of +/- 5%. The temperature sensor measures from 0 to 50 degrees Celsius with an accuracy of +/- 2%. The sensor will relay the environmental temperature and humidity data at user specified intervals.

5.3. User Interface Subsystem Requirements and Specifications

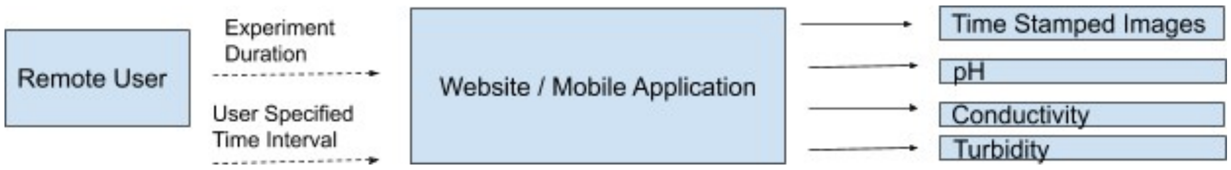


Figure 4 - Level 2 Block Diagram Subsystem 3: User Interface

<u>User Interface subsystem requirements</u>

- The data must be easily accessible to a remote user.

<u>User Interface subsystem specifications</u>

- The user must be able to access all experiment data from the system.

5.4. Power Management Subsystem Requirements and Specifications

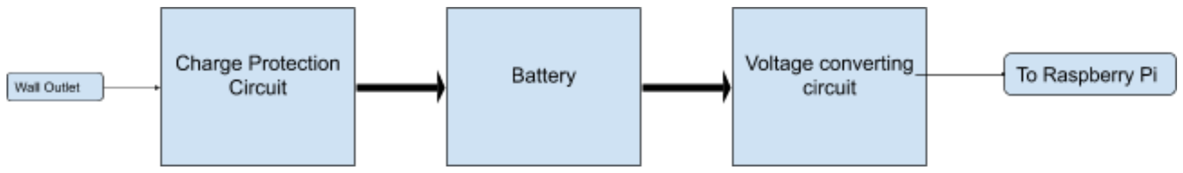


Figure 5 - Level 2 Block Diagram Subsystem 4: Power Management

<u>Power management subsystem requirements</u>

- System supplies voltage to Raspberry Pi.
- System charges the battery.

<u>Power management subsystem specifications</u>

- The Raspberry Pi takes 5 Volts.
- A wall outlet will charge the battery.

5.4.1. Charge Protection Circuit Requirements and Specifications

<u>Charge protection circuit requirements</u>

- Circuit optimizes the safe charging of the battery.

<u>Charge protection circuit specifications</u>

- The charge protection circuit will stop charging the battery at 100% charge.
- The charge protection circuit will monitor the charge level in each cell and keep each cell at the same level of charge.

5.4.2. Battery Requirements and Specifications

<u>Battery Requirements</u>

- The battery will supply power to the Raspberry Pi.

<u>Battery Specifications</u>

- The battery will charge from the charge protection circuit.
- The battery will output 12 Volts, which it will supply to the voltage conversion circuit.
- The battery will hold at least 200ah of charge

5.4.3. Voltage Converter Circuit Requirements and Specifications

<u>Voltage converter circuit Requirements</u>

- Circuit converts battery output to a lower voltage to supply Raspberry Pi.

Voltage converter Specifications

- The voltage converter will take 7V from the output of the battery and output 5V to power the Raspberry Pi.

6. Module Design (All)

6.1. Raspberry Pi Subsystem

The Raspberry Pi is the central computation and storage module for the system. It receives power from the power management module and receives input from programmatic user instructions. It runs experimental scripts and gathers data via input from attached sensors. It stores and exports data via Rsync / GUI / local storage.

A Raspberry Pi was chosen for this subsystem over other alternatives such as Beagleboard and Arduino. The Raspberry Pi was chosen for its low cost and its ability to meet the other specifications required by the project.

6.2. Sensor Interface Subsystem

The sensor module works via the Raspberry Pi's GPIO header outputting programmatic instructions. Temperature and humidity are measured via sensor straight from this header, while all Atlas sensor data is taken through conversion to I2C over BNC via the Tentacle T3 Hat. In addition, the Pi Camera is powered from another GPIO slot on board. All received data from the sensors and camera are then stored locally and manipulated by programmatic instruction.

6.2.1. Camera

The Raspberry Pi camera was chosen due to the ease of connecting it to the Raspberry Pi. Other cameras were investigated, such as ones connected by USB, but none met the project needs.

6.2.2. Tentacle Hat

This module was chosen based on the choice of sensors. This module exists because of the following two sensors, so no alternatives were considered.

6.2.3. Conductivity Sensor

This sensor was chosen in conjunction with the client. Alternatives were not considered.

6.2.4. pH Sensor

This sensor was chosen in conjunction with the client. Alternatives were not considered.

6.2.5. Temperature and Humidity Sensor

Initially, a simple thermistor and different off-the-shelf humidity sensors were investigated. We decided on the Adafruit DHT1 due to its ease of use combining both sensors into one and the ease of connecting it to the system.

6.3. User Interface Subsystem

Input for the UI system consists only of data and information about each experiment being related remotely from central Raspberry Pi. The UI receives this information and presents it to the user visually and enables the user to interact with and download that data.

The UI will receive this information in real time via webserver file transfer.

6.4. Power Management Subsystem

This module supplies power to the rest of the system. This module was designed with the need for the system to be able to operate in spaces with no wall power. The charge protection circuit receives power from the wall and distributes it evenly across all cells in the battery, stopping when the cells are full. The charge protection circuit regulates the output of the battery, preventing them from being fully discharged. The voltage converter changes the voltage outputted by the battery to the 5V required by the Raspberry Pi.

6.4.1. Charge Protection Circuit

No alternatives have been considered for this module. The system requires a battery to run and needs a way to safely charge.

6.4.2. Battery

No alternatives have been considered for this module. A battery is a requirement for our system to run. A power test was not conducted due to the restrictions of the pandemic.

6.4.3. Voltage Converter Circuit

No alternatives have been considered for this module. The Raspberry Pi requires 5V to run and a voltage converter is needed to regulate the voltage from the battery.

7. Design Changes Since FDRF

Since the completion of the FDR, design changes have primarily been related to the data pipeline structure for this project. These changes were born out of a necessity to facilitate user interaction with this system, as well as to provide redundant options for data transfer and storage. In order to complete both of these tasks, a Redis queue was added for chronning data transfer jobs and keeping them active in the event of some kind of failure, and a remote data server was added for storage and organization.

At the current time, gathered data is collected and stored in labeled local directories on the Raspberry Pi. All relevant timestamps, sensor data, and image names are then queued by Redis and transferred to a SQL Server database running on the server. For the sake of redundancy, all relevant Raspberry Pi local files are then synced to a directory on this server. Redis allows for consistency in transfer and maintains the operating status of said transfer jobs, and the remote data server allows for easy data presentation by the front end and acts as a secondary storage location.

It is important to note that other than these changes, the primary structure and design for this project has remained consistent. Though completion of work related to the front end and battery circuit was hindered extremely by the current state of the Coronavirus outbreak, these portions of the project are still integral to its functionality and purpose. In order to craft a successful end-to-end product, these sections need to be implemented.

8. Module and System Tests

Note: while the tests discussed below were supposed to be run, due to the current Coronavirus outbreak affecting certain aspects of this project's completion, alternate testing strategies were devised. An overall system test was performed with varying parameters for overall time and sensor reading interval. Our team felt that this was a productive testing plan to complete what could be done with the current situation.

The three tests devised to test the overall system, sensor accuracy over time, data transfer behavior, and system behavior under load are as follows:

- 1 hour test – 360 sensor readings
- 2.5 hour test – 50 sensor readings
- 8 hour test – 97 sensor readings

8.1. Raspberry Pi Subsystem Tests

Tests:

- Pi Stress Test - Run a worst-case scenario experiment and check that the Raspberry Pi continuously sends the data to the user and storage solution.
- Pi / Remote Storage Stress Test - Run alternate test case scenario with large set of data; and ensure the Raspberry Pi is able to complete a run of transferring multiple megabyte files continuously.

Specifications:

- The Raspberry Pi will take user input and use that to collect data at the specified time.
- The Raspberry Pi will process the data collected and send that data to the user interface via rsync.

8.2. Sensor Subsystem Tests

Tests:

- Stress Test - Run a worst-case scenario experiment (4 weeks sampling every 2 seconds) and check that the sensors collect data over the whole time period.
- Use sensors to measure known variables (such as a known room temperature or standard pH solution) to check accuracy.

Specifications:

- The Raspberry Pi GPIO controller will translate the programmatic read instructions to the sensors.
- The LED will inform the user of each time the system takes a reading.
- The Raspberry Pi camera will take a photo at each read interval.
- The Raspberry Pi Tentacle T3 hat will relay the programmatic instructions to the Atlas sensors.
- The sensors will relay environmental condition data at each specified time interval.

8.3. User Interface Subsystem Tests

Tests:

- Test functionality of visually interactive components by activating them and observing outputs.
- Verify the application's responsiveness on screen sizes for tablets (width: 1024/768 pixels), monitors (width: infinite pixels), and mobile devices (width: 480/320 pixels).

Specifications:

- The user must be able to access and observe current system status.
- The user must be able to access and observe all sensor readings and pictures taken over the course of an entire experiment.

Note: Due to the Coronavirus outbreak not all the User Interface tests were completed. In an ideal world where there is more time these tests should still be done.

8.4. Power Management Subsystem Tests

Tests:

- Verify that the maximum output of the subsystem is 5 volts to the Raspberry Pi and any attached equipment.
- Run a worst-case scenario experiment (4 weeks sampling every 2 seconds) and check that power is applied at appropriate levels for the entirety of the experiment.
- Monitor the shape of the battery during charging and system operation and observe any alterations that occur.

Specifications:

- The system will supply 5 volts to the Raspberry Pi and any attached equipment.
- The battery must last for the maximum possible duration of an experiment; for this project, that length is anticipated to be a maximum of 4 weeks.
- The charge protection circuit will stop charging the battery when it has reached 100% charge.

9. Applicable Standards (All)

9.1. Probes

NSF/ANSI-51 Section 4.1

Pi / Remote Storage Stress Test - Run alternate test case scenario with large set of data; and ensure the Raspberry Pi can complete a run of transferring multiple megabyte files continuously.

NSF/ANSI-51 Section 7.6.1

Glass and glass-like materials. Such materials shall not be used for direct food contact that are also subject to impact by hard objects during use such as countertops, tabletops, cutting boards, cooking surfaces, and food preparation utensils. This standard has the same use case as the previous one.

73/23/EEC

The aim of this directive is to ensure that electrical equipment cannot be placed on the market if, when installed and maintained, it endangers the safety of persons, domestic animals or property. This standard is for the probes and it ensures that there is no danger due to being electrocuted.

ISO/IEC 17025

ISO/IEC 17025 enables laboratories to demonstrate that they operate competently and generate valid results, thereby promoting confidence in their work both nationally and around the world. It also helps facilitate cooperation between laboratories and other bodies by generating wider acceptance of results between countries. This standard is applicable to this project since our product will be used in labs to gather results.

9.2. Raspberry Pi

EC 60950-1:2005

Mains and battery-powered general low voltage electrical equipment must follow safety standards related to temperature exposure, dust and moisture exposure, etc. This standard applies to operation of RaspberryPi 3B+.

EN 55022 CISPR 22

There are radio disturbance characteristic limits and methods of measurement for electrical equipment. This establishes uniform requirements for radio disturbance, standard operating conditions, and standard interpretations of testing results. It applies to all standard and networked operations of RaspberryPi 3B+.

EN 55024 CISPR 24

Tests and places standardized limitations for device electrostatic discharge, surges, power frequency magnetic fields, power interruptions, and RF interference.

Covers standard and networked operations of RaspberryPi 3B+ and all attached electrical equipment.

EN 300 328 V2.1.1

Wideband transmission systems; data transmission equipment operating in the 2.4 GHz ISM band using wide band modulation techniques. Limitations on output RF power and behavioral maintenance of networked equipment. Applies to networked operation of RaspberryPi 3B+.

9.3. System

IEC 62311:2019

Covers appropriate maintenance of equipment-generated exposure to electric, magnetic, and electromagnetic for devices operating from 0 Hz to 300 GHz. Standard limitations are applied to equipment via localized standards, applicable to this standard. Applies to the operation of full system.

EN 55032 CISPR 32

Procedural standard for limiting levels of spurious signals generated by multimedia equipment for electromagnetic protection in industrial, commercial, and residential environments. Applies to the operation of full system.

9.4. Hardware

ISO 4032:2012

Hexagon regular nuts (style 1) — Product grades A and B. Used in sizing holes for nuts and bolts to assemble enclosure.

9.5. Protocols

UM10204

Philips Semiconductors (now NXP Semiconductors) developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. This standard is used for communicating between the Raspberry Pi and the sensors.

RFC 4256

The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network. This document describes a general-purpose authentication method for the SSH protocol, suitable for interactive authentications where the authentication data should be entered via a keyboard. This standard is useful to our project since it is how we are going about sending data wirelessly using Rsync.

10. Summary and Conclusions (R)

This semester, all major tasks with the sensors were completed; sensors were integrated fully with the back-end code. In addition to this, a front-end GUI and enclosure modeling were drafted in preparation for full system operation. On the front-end the user can fully see all the data as opposed to having to look in the database now. Final drawings for the enclosure were done, however not everything was printed due to the current situation. The system is ready to be tested for an experiment if time would allow.

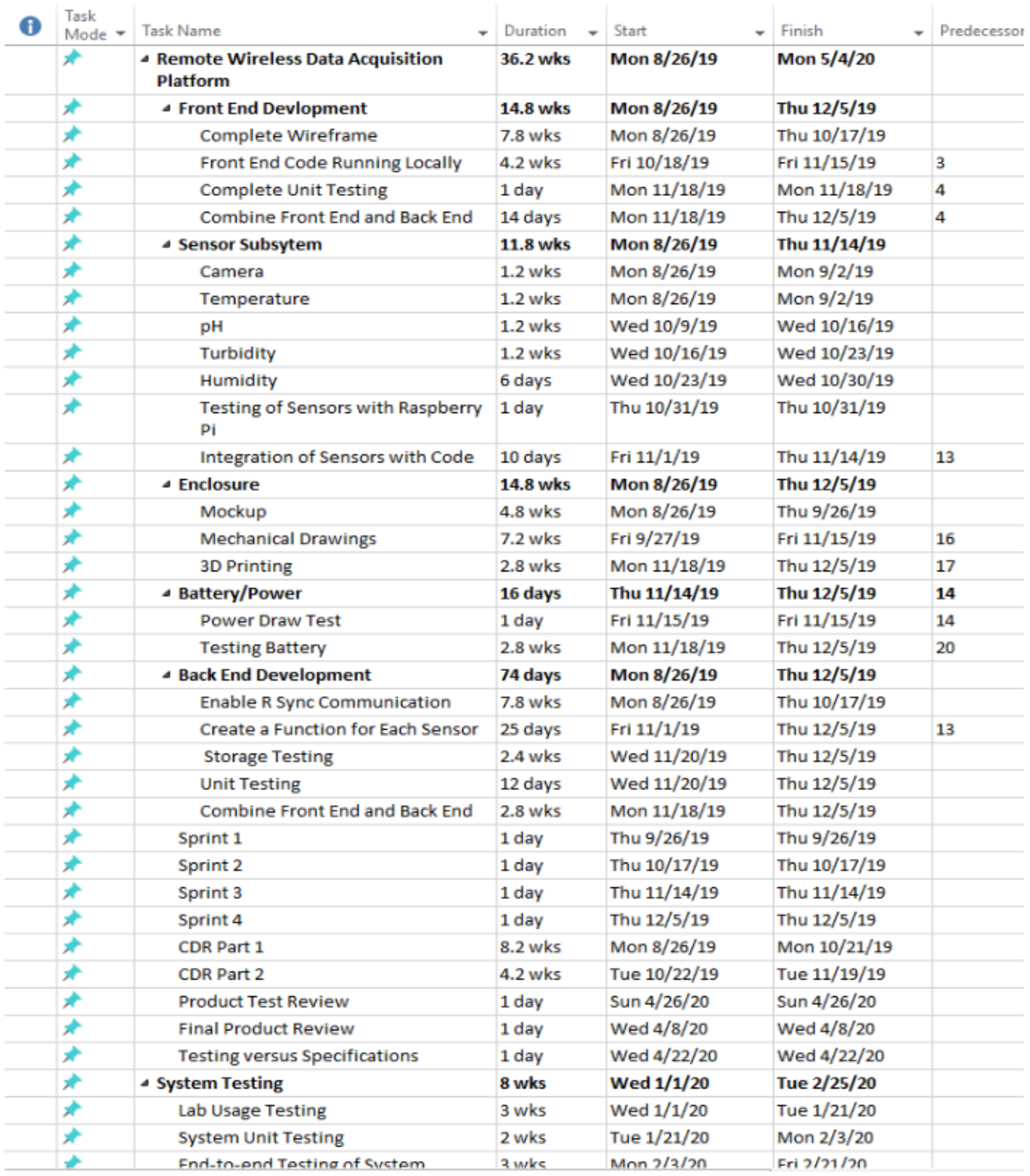
This semester, several improvements were made to the overall efficiency and functionality of the system's data pipeline. A Redis queue was added to monitor and schedule the submission of data to the project's newly added data server. This server was deployed into a redundant storage scheme where all information stored locally on the Raspberry Pi is then subsequently transferred to it via Rsync. In addition to this, a SQL Server was deployed on this remote storage space to be implemented by the graphical interface. This SQL database serves not only as an efficient way to serve the information to a user over the internet, but also as a quick lookup table to ensure that the system is functioning properly via system diagnostics read at every sensor interval. Overall, the last semester has yielded great productivity in the track to completion of this project. The data pipeline is fully prepared to be deployed both online and as a storage solution for a wider-spread audience.

11. References

Unlock the Power of Pay. (n.d.). Retrieved from https://www.salary.com/

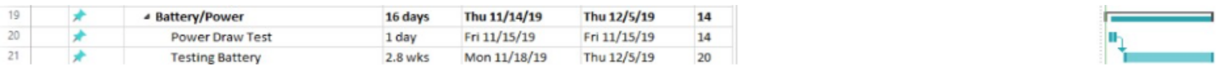
12. Appendices

12.1. Timeline Estimation and Milestones



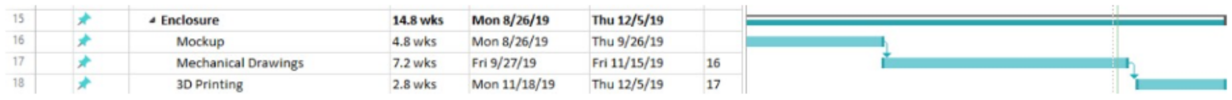
	Task Mode	Task Name	Duration	Start	Finish	Predecessor
	📌	**Remote Wireless Data Acquisition Platform**	**36.2 wks**	**Mon 8/26/19**	**Mon 5/4/20**	
	📌	**Front End Devlopment**	**14.8 wks**	**Mon 8/26/19**	**Thu 12/5/19**	
	📌	Complete Wireframe	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	📌	Front End Code Running Locally	4.2 wks	Fri 10/18/19	Fri 11/15/19	3
	📌	Complete Unit Testing	1 day	Mon 11/18/19	Mon 11/18/19	4
	📌	Combine Front End and Back End	14 days	Mon 11/18/19	Thu 12/5/19	4
	📌	**Sensor Subsytem**	**11.8 wks**	**Mon 8/26/19**	**Thu 11/14/19**	
	📌	Camera	1.2 wks	Mon 8/26/19	Mon 9/2/19	
	📌	Temperature	1.2 wks	Mon 8/26/19	Mon 9/2/19	
	📌	pH	1.2 wks	Wed 10/9/19	Wed 10/16/19	
	📌	Turbidity	1.2 wks	Wed 10/16/19	Wed 10/23/19	
	📌	Humidity	6 days	Wed 10/23/19	Wed 10/30/19	
	📌	Testing of Sensors with Raspberry Pi	1 day	Thu 10/31/19	Thu 10/31/19	
	📌	Integration of Sensors with Code	10 days	Fri 11/1/19	Thu 11/14/19	13
	📌	**Enclosure**	**14.8 wks**	**Mon 8/26/19**	**Thu 12/5/19**	
	📌	Mockup	4.8 wks	Mon 8/26/19	Thu 9/26/19	
	📌	Mechanical Drawings	7.2 wks	Fri 9/27/19	Fri 11/15/19	16
	📌	3D Printing	2.8 wks	Mon 11/18/19	Thu 12/5/19	17
	📌	**Battery/Power**	**16 days**	**Thu 11/14/19**	**Thu 12/5/19**	**14**
	📌	Power Draw Test	1 day	Fri 11/15/19	Fri 11/15/19	14
	📌	Testing Battery	2.8 wks	Mon 11/18/19	Thu 12/5/19	20
	📌	**Back End Development**	**74 days**	**Mon 8/26/19**	**Thu 12/5/19**	
	📌	Enable R Sync Communication	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	📌	Create a Function for Each Sensor	25 days	Fri 11/1/19	Thu 12/5/19	13
	📌	Storage Testing	2.4 wks	Wed 11/20/19	Thu 12/5/19	
	📌	Unit Testing	12 days	Wed 11/20/19	Thu 12/5/19	
	📌	Combine Front End and Back End	2.8 wks	Mon 11/18/19	Thu 12/5/19	
	📌	Sprint 1	1 day	Thu 9/26/19	Thu 9/26/19	
	📌	Sprint 2	1 day	Thu 10/17/19	Thu 10/17/19	
	📌	Sprint 3	1 day	Thu 11/14/19	Thu 11/14/19	
	📌	Sprint 4	1 day	Thu 12/5/19	Thu 12/5/19	
	📌	CDR Part 1	8.2 wks	Mon 8/26/19	Mon 10/21/19	
	📌	CDR Part 2	4.2 wks	Tue 10/22/19	Tue 11/19/19	
	📌	Product Test Review	1 day	Sun 4/26/20	Sun 4/26/20	
	📌	Final Product Review	1 day	Wed 4/8/20	Wed 4/8/20	
	📌	Testing versus Specifications	1 day	Wed 4/22/20	Wed 4/22/20	
	📌	**System Testing**	**8 wks**	**Wed 1/1/20**	**Tue 2/25/20**	
	📌	Lab Usage Testing	3 wks	Wed 1/1/20	Tue 1/21/20	
	📌	System Unit Testing	2 wks	Tue 1/21/20	Mon 2/3/20	
	📌	End-to-end Testing of System	3 wks	Mon 2/3/20	Fri 2/21/20	

Figure 6 – Overall Gantt Chart



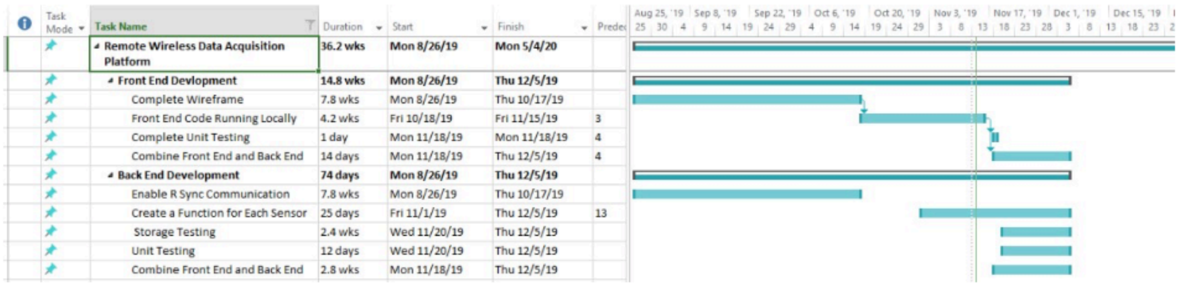
		Task Name	Duration	Start	Finish	Predecessors
19	📌	**Battery/Power**	**16 days**	**Thu 11/14/19**	**Thu 12/5/19**	**14**
20	📌	Power Draw Test	1 day	Fri 11/15/19	Fri 11/15/19	14
21	📌	Testing Battery	2.8 wks	Mon 11/18/19	Thu 12/5/19	20

Figure 7 – Battery Subsystem Gantt Chart



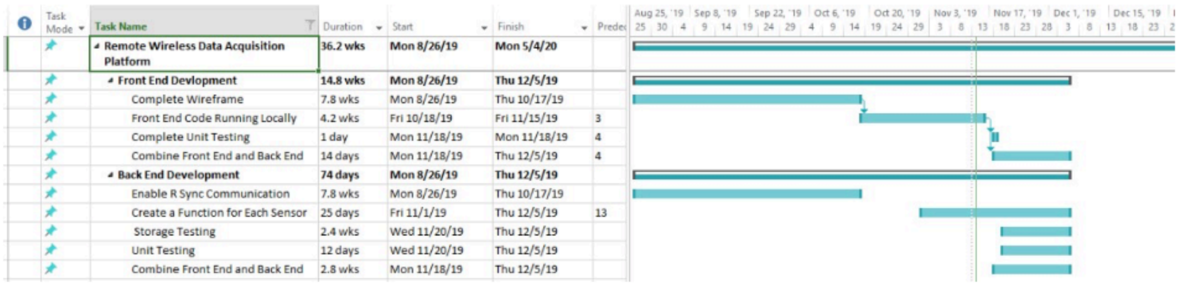
		Task Name	Duration	Start	Finish	Predecessors
15	📌	**Enclosure**	**14.8 wks**	**Mon 8/26/19**	**Thu 12/5/19**	
16	📌	Mockup	4.8 wks	Mon 8/26/19	Thu 9/26/19	
17	📌	Mechanical Drawings	7.2 wks	Fri 9/27/19	Fri 11/15/19	16
18	📌	3D Printing	2.8 wks	Mon 11/18/19	Thu 12/5/19	17

Figure 8 – Enclosure Gantt Chart



Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Remote Wireless Data Acquisition Platform	**36.2 wks**	**Mon 8/26/19**	**Mon 5/4/20**	
	Front End Devlopment	**14.8 wks**	**Mon 8/26/19**	**Thu 12/5/19**	
	Complete Wireframe	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	Front End Code Running Locally	4.2 wks	Fri 10/18/19	Fri 11/15/19	3
	Complete Unit Testing	1 day	Mon 11/18/19	Mon 11/18/19	4
	Combine Front End and Back End	14 days	Mon 11/18/19	Thu 12/5/19	4
	Back End Development	**74 days**	**Mon 8/26/19**	**Thu 12/5/19**	
	Enable R Sync Communication	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	Create a Function for Each Sensor	25 days	Fri 11/1/19	Thu 12/5/19	13
	Storage Testing	2.4 wks	Wed 11/20/19	Thu 12/5/19	
	Unit Testing	12 days	Wed 11/20/19	Thu 12/5/19	
	Combine Front End and Back End	2.8 wks	Mon 11/18/19	Thu 12/5/19	

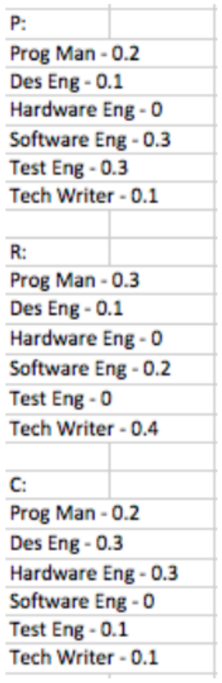
Figure 9 – Front/Back End Gantt Chart



Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Remote Wireless Data Acquisition Platform	**36.2 wks**	**Mon 8/26/19**	**Mon 5/4/20**	
	Front End Devlopment	**14.8 wks**	**Mon 8/26/19**	**Thu 12/5/19**	
	Complete Wireframe	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	Front End Code Running Locally	4.2 wks	Fri 10/18/19	Fri 11/15/19	3
	Complete Unit Testing	1 day	Mon 11/18/19	Mon 11/18/19	4
	Combine Front End and Back End	14 days	Mon 11/18/19	Thu 12/5/19	4
	Back End Development	**74 days**	**Mon 8/26/19**	**Thu 12/5/19**	
	Enable R Sync Communication	7.8 wks	Mon 8/26/19	Thu 10/17/19	
	Create a Function for Each Sensor	25 days	Fri 11/1/19	Thu 12/5/19	13
	Storage Testing	2.4 wks	Wed 11/20/19	Thu 12/5/19	
	Unit Testing	12 days	Wed 11/20/19	Thu 12/5/19	
	Combine Front End and Back End	2.8 wks	Mon 11/18/19	Thu 12/5/19	

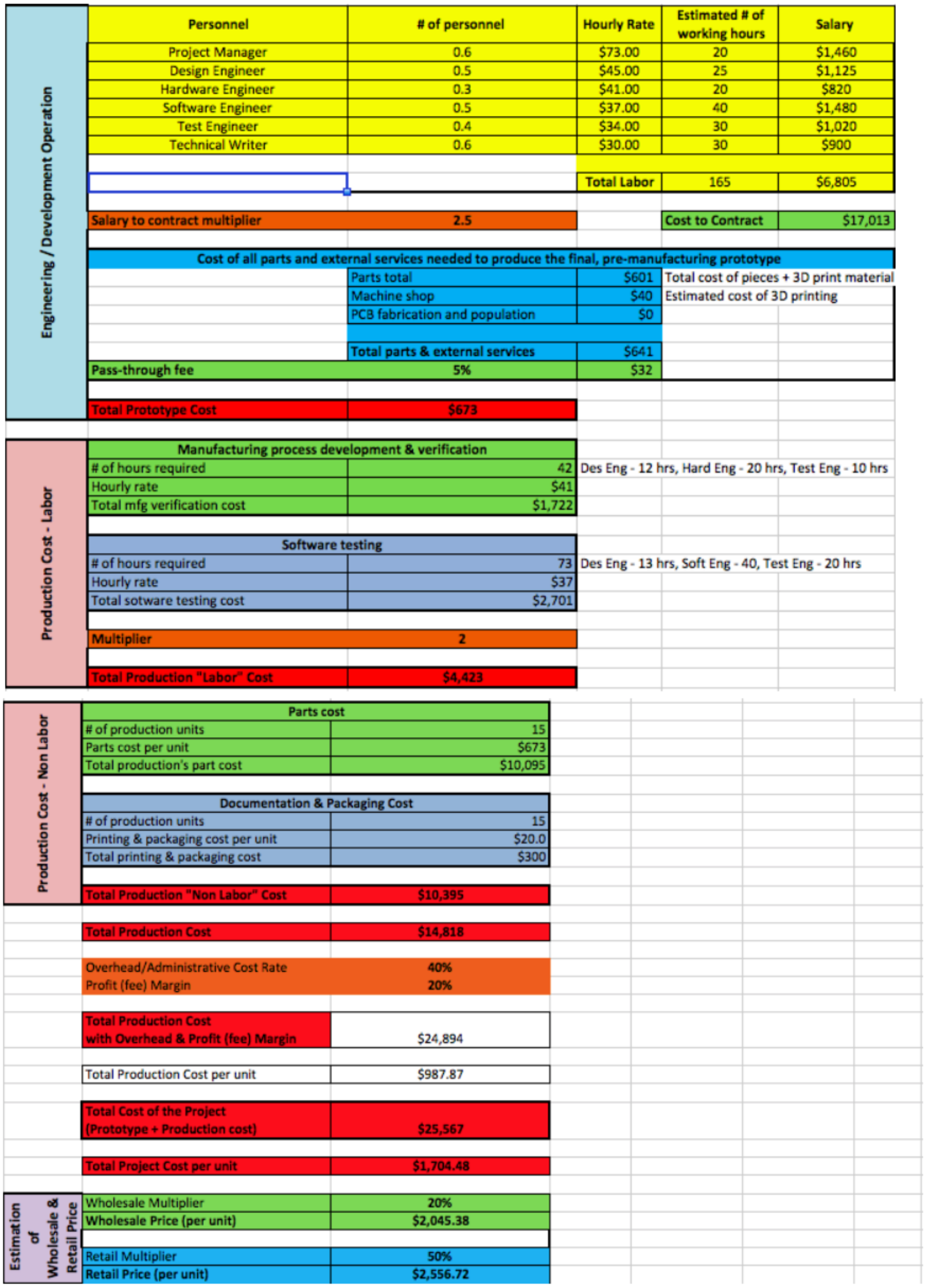
Figure 10 – Sensor Subsystem Gantt Chart

12.2. Economic Analysis



P:	
Prog Man - 0.2	
Des Eng - 0.1	
Hardware Eng - 0	
Software Eng - 0.3	
Test Eng - 0.3	
Tech Writer - 0.1	
R:	
Prog Man - 0.3	
Des Eng - 0.1	
Hardware Eng - 0	
Software Eng - 0.2	
Test Eng - 0	
Tech Writer - 0.4	
C:	
Prog Man - 0.2	
Des Eng - 0.3	
Hardware Eng - 0.3	
Software Eng - 0	
Test Eng - 0.1	
Tech Writer - 0.1	

Figure 11 – Working Hours per Person



Engineering / Development Operation

Personnel	# of personnel	Hourly Rate	Estimated # of working hours	Salary
Project Manager	0.6	$73.00	20	$1,460
Design Engineer	0.5	$45.00	25	$1,125
Hardware Engineer	0.3	$41.00	20	$820
Software Engineer	0.5	$37.00	40	$1,480
Test Engineer	0.4	$34.00	30	$1,020
Technical Writer	0.6	$30.00	30	$900
		Total Labor	165	$6,805
Salary to contract multiplier	2.5		Cost to Contract	$17,013

Cost of all parts and external services needed to produce the final, pre-manufacturing prototype			
	Parts total	$601	Total cost of pieces + 3D print material
	Machine shop	$40	Estimated cost of 3D printing
	PCB fabrication and population	$0	
	Total parts & external services	$641	
Pass-through fee	5%	$32	
Total Prototype Cost	$673		

Production Cost - Labor

Manufacturing process development & verification		
# of hours required	42	Des Eng - 12 hrs, Hard Eng - 20 hrs, Test Eng - 10 hrs
Hourly rate	$41	
Total mfg verification cost	$1,722	

Software testing		
# of hours required	73	Des Eng - 13 hrs, Soft Eng - 40, Test Eng - 20 hrs
Hourly rate	$37	
Total sotware testing cost	$2,701	
Multiplier	2	
Total Production "Labor" Cost	$4,423	

Production Cost - Non Labor

Parts cost	
# of production units	15
Parts cost per unit	$673
Total production's part cost	$10,095

Documentation & Packaging Cost	
# of production units	15
Printing & packaging cost per unit	$20.0
Total printing & packaging cost	$300
Total Production "Non Labor" Cost	$10,395

Total Production Cost	$14,818
Overhead/Administrative Cost Rate	40%
Profit (fee) Margin	20%
Total Production Cost with Overhead & Profit (fee) Margin	$24,894
Total Production Cost per unit	$987.87
Total Cost of the Project (Prototype + Production cost)	$25,567
Total Project Cost per unit	$1,704.48

Estimation of Wholesale & Retail Price

Wholesale Multiplier	20%
Wholesale Price (per unit)	$2,045.38
Retail Multiplier	50%
Retail Price (per unit)	$2,556.72

Figure 12 – Economic Analysis Breakdown

This product would be offered as a good, with all the desired sensors and attachments wrapped in with the cost of the Raspberry Pi and the software application. The retail price of this product would be $2,556.72, as can be seen in the figure above. An additional option to use this software as a standalone option could also be released, where the user would provide their own Raspberry Pi and sensors, configuring the device on their own.

12.3. Part List

Part Name	Description	Manufacturer	Manufacturer's Part #	Supplier	Cost ($)	Quantity
Single Board Computer	Raspberry Pi 3B +	Raspberry Pi	3775	Adafruit	35.00	1
GPIO Extender	RPi GPIO Breakout Expansion Board & Ribbon	Arceli	ZB2L3	Amazon	7.99	1
Breadboard	Solderless Breadboard Kit	REXQualis	RQ-BK-002	Amazon	9.86	1
Resistor	220 Ohm	E-Projects	32121600	Amazon	6.00	1 (100 resistors)
Temperature-Humidity Sensor	DHT11	Adafruit	100502	Amazon	9.61	1
LED	Red Diffused LED	Jameco	2152155	Jameco	0.59	5

Conductivity Sensor	Conductivity K 1.0 Kit	Atlas Scientific	EC-KIT-1.0	Atlas Scientific	215.00	1
pH Sensor	pH Kit	Atlas Scientific	KIT-101P	Atlas Scientific	164.00	1
GPIO Hat	Tentacle T3	Whitebox Labs for Atlas Scientific	TEN-T3	Atlas Scientific	105.00	1
Camera	RPi Camera Module V2-8 MP	Raspberry Pi	RPI-CAM-V2	Amazon	24.80	1
Ethernet Cable	CAT 6 Ethernet Cable	Mediabridge	31-299-10B	Amazon	5.84	1

12.4. Qualifications of Key Personnel

Rachel Zaltz's qualifications make her a stellar candidate to work on this project. Her course work at the GWU includes Operating Systems, Microprocessors, Embedded Systems, Intro to C Programming will aid her in this project. [DZ2] Rachel's wide knowledge of different programming languages allow her to adapt quickly. Her internship experience includes an extensive amount of front-end development, which will be useful towards the end of the project in building a user- friendly interface to view all the data.

Connor Kraft's qualifications allow him to be a good team member. His work teaching children STEM classes over the summer, including teaching Python and Raspberry Pi courses, will aid in the overall design and troubleshooting of the project. Further, the teaching of 3D design and printing will be useful towards the end of the project when building an enclosure. His coursework at GWU includes Microprocessors, Embedded systems, and Circuit theory will all help in the hardware aspect of this project.

Phillip Jones has numerous experiences that are directly related to this project that will greatly facilitate with him making contributions. His coursework at GWU includes several Electronics Design classes, Microprocessor Systems, Introduction to C Programming, and Circuit Design. Phillip has worked for two semesters with SEAS computing as a Raspberry Pi student workshop director and has additional supplementary work experience with the same company. Lastly, he has extensive programming experience working on an automated data pipeline web application for Gamma Ray Burst behavioral analysis.

12.5. Intellectual Contributions

1. The group will purchase all sensors, power supply, and any other necessary fabrication materials.
    a. The group will use these to fabricate all the above modules.
    b. Sensors will be tested and calibrated for accuracy
2. We will design, implement, and test code to gather the sensor data.
3. We will implement Rsync to access the previously mentioned data wirelessly.
    a. We will test this system with a wide variety of file sizes and file amounts
4. We will design, implement, and test a GUI that interacts with the previously mentioned code.

12.6. Teaming Arrangements

Throughout this project, all team members will work as a cohesive unit to ensure the product is designed properly. Each team member will lead one part of the project:

- Phillip will lead the group in implementing the physical code on the hardware.
- Connor will oversee testing the code and evaluating the results. He will also design an enclosure to house the final project.
- Rachel will build the GUI to display the data. She will also facilitate all documentation throughout the duration of the project.

12.7. User Manual

Setup Steps:

Image the Raspberry Pi:

- The official guide for this process can be found here: https://www.raspberrypi.org/documentation/installation/installing-images/
- For the purposes of this generation of the product, our team used Raspbian GNU / Linux 9, also known as Stretch, but the OS version for the system should have little influence on proper operation

Connect the Pi to basic peripherals and boot:

- Connect mouse and keyboard via USB
- Connect HDMI cable to external monitor
- Connect Ethernet cable for wired network connection (optional, only needed for setup)
- Connect the Pi to wall power adapter
- Boot into Raspbian or preferred OS

Enable system interfaces:

- Once booted in, click on the Raspberry Pi symbol to open the system menu
- Navigate down to “Preferences” and open the “Raspberry Pi Configuration” tab
- Navigate to the “Interfaces” tab and ensure that Camera, SSH, VNC, and I2C are enabled

Setup remote monitoring webserver on the Pi:

- Pagekite is used to assign hostnames to outward facing VNC addresses to make remote monitoring possible from anywhere
- Sign up for Pagekite here: https://pagekite.net/signup/
- Follow the guide at the following page: https://pagekite.net/support/quickstart/

- Once initial setup is completed, two kites should be added by executing the following from command line:
    - "python2 pagekite.py --add 80 <site_name>.pagekite.me"
    - "python2 pagekite.py --add 5900 raw:vnc.<site_name>.pagekite.me:5900"
- Both of these hostnames need to be up at all times and, after their creation, can be launched using the following from command line:
    - "python2 pagekite.py 80 <site_name>.pagekite.me"
    - "python2 pagekite.py port# raw:vnc.<site_name>.pagekite.me:5900"
- This application's website contains instructions on how to run this program from startup, if desired

Setup VNC monitoring on remote machine:

- Download VNC Viewer by RealVNC on the machine the Pi will be monitored from
- Navigate to preferences and locate the "Proxy" tab
- Select the options to "Use these proxy settings"
- Enter the following values:
    - Server: vnc.<site_name>.pagekite.me:443
    - Type: HTTP Connect
    - User / Pass: pi / raspberry (initial image user / pass)
- You can then connect via VNC to the Pi via the address vnc.<site_name>.pagekite.me:5900

Share SSH keys between the Pi and data server:

- From the Raspberry Pi, generate an SSH key by executing the following on command line:
    - "ssh-keygen"
- Then copy this key to the remote data server by executing the following:
    - "ssh-copy-id –i ~/.ssh/id_rsa.pub <X.X.X.X>"
- This step is absolutely vital for this application's performance and can be tested by attempting to SSH to the remote data server without password:
    - "ssh <X.X.X.X>"

Setup Redis-Server on the Pi:

- Redis-Server is used implement a job queuing process processing data from the application
- The setup process can be found at the following: https://habilisbest.com/install-redis-on-your-raspberrypi
- Once Redis is properly setup, this queue webserver should be run constantly when data is being processed by doing the following from command line:
    - "redis-server"

Setup SQL Server on data server:

- Both SQL Server and the SQL Command Line Tools should be installed on the data server
- The following process should be followed:
    - https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-ubuntu?view=sql-server-ver15
    - Please note, an Ubuntu based server is currently deployed for this group, but Microsoft provides installation guides for these tools for most common operating systems
- Once SQL Server is properly setup, a table with the following columns and data types should be created
    - Timestamp – VARCHAR

  - pH – FLOAT
  - Conductivity – FLOAT
  - Temperature – FLOAT
  - Humidity – FLOAT
  - Image Name – VARCHAR

Setup ImageMagick on data server (optional):

- ImageMagick can be used to view images via X server when a machine establishes an SSH connection with the data server
- The following shows the setup process for a typical Ubuntu machine: https://linuxconfig.org/how-to-install-imagemagick-7-on-ubuntu-18-04-linux
- In order to launch the X server on startup of the SSH session, simply type the following when first connecting:
  - "ssh <user>@<X.X.X.X> -X"
- Then, in order to display any image type the following on command line, once inside the image's directory:
  - "display <filename>.<extension>"
- If continual issues occur with displaying images, make sure the $DISPLAY environment variable is set correctly on your data server by executing the following:
  - "export DISPLAY=localhost:0.0"

Setup full peripherals:

- Connect the PiCamera via GPIO ribbon the Pi's camera port
- Connect the Atlas T3 Tentacle hat to the Pi's GPIO header and attach the appropriate Atlas sensors to their respective serial cable header
- Attach the GPIO ribbon cable to the T3's GPIO header extension, then connect a breadboard on the ribbon cable
- Connect all additional sensors to their respective GPIO ports on the ribbon cable's breadboard connector
- Attach battery powering system, lower device into enclosure and properly attach sensors and camera to arms

Download application package from Github:

- Download full application (3 scripts) from Github and create an appropriate environment for them (i.e. create and name a folder solely for holding the scripts)
- Establish a folder called "capstone" on the Pi's desktop, which will be used for storing the processed data and images

Download Python packages for scripts:

- Execute the following from the command line to install a Python package:
  - "pip install <pkg_name>"
- The following is a list and description of all Python packages needed to run this application:
  - time – Used to implement wait periods, timings
  - sys – Used to force clean exit from script on error outs
  - OS – Used to execute CMD commands from inside script
  - subprocess – Used to call CMD subprocesses from inside script (i.e. rsync calls)
  - picamera – Used to operate and manipulate Pi camera
  - datetime – Used to establish time stamped naming conventions
  - RPi.GPIO – Used to operate Pi GPIO ports

- Adafruit_DHT – Used to gather temperature and humidity data from Adafruit DHT11 or DHT22 sensor
- IO – Used to create and implement filestreams
- fcntl – Used to interact with I2C parameters for Atlas sensors
- string – Used to interact with and manipulate strings
- copy – Used to move I2C parameters between addresses by copying values under certain specified conditions
- RQ – Used to simply implement a Redis job queue (note: this is unofficial)
- Redis – Used to connect to Redis queue from inside script (note: this is official)
- Pyodc – Used to setup script connection to SQL Server

Edit variable information in scripts:

- atlas.py – queue name for Redis queue, username and IP for data server, rsync folder name on data server
- atlas_upload.py – queue name for Redis queue, connection details for SQL Server on data server, schema and table names for SQL Server table

Script details:

- atlas.py:
  - Takes sensor readings and images, submits data to Redis queue and uploads CSV and images to directory on data server
  - Run at user's discretion (i.e. run when user wants to run an experiment)
  - Execute with the following: "python atlas.py"
- atlas_worker.py:
  - Acts as middleman and takes enqueued data to be submitted to queue and pushed to the correct script
  - This script can be left running at all times due to low CPU and memory impact, as it will process results any time they are submitted
  - Execute with the following: "python atlas_worker.py <queue_name>"
- atlas_upload.py:
  - Uploads the queued data to the remote data server's SQL Server table
  - NEVER interacted with by the user, this script is only used when called from the main atlas.py

<u>System Operational Notes:</u>

- Python version to be used – 2.7.13
  - Though Python 2.7 is going to be considered fully deprecated soon, the only way that the current build works with Atlas' provided I2C software is if it is used
  - NOTE: ensure that the relevant version of Pip is deployed for installing packages for the correct version of Python
- Constantly running scripts
  - The easiest way to implement the technologies discussed in this guide without too much maintenance is to have several scripts continually run (or to add them to your Bash configuration for launch on startup)
  - It is recommended that both Pagekite addresses (main and VNC), atlas_upload, and redis-server stay continually running
    - This enables the application to run with end-to-end success by only executing the main atlas.py

Front End Code Notes:

- This project is uses a React front end app that is written in JavaScript.
- Currently the code can be seen on a front-end GUI. However, due to some impediments related to the Coronavirus it is very basic and there is no user experience.
- Information and documentation for React can be found here: https://reactjs.org/
- The connection to the front-end is made in the Index.js file
- To setup the environment for into the terminal and run the following:
    - mkdir "folder name"
    - touch package.json
    - npm init -y
    - Insert the code into the index.js file
    - Install the following packages:
        - yarn add express
        - yard add mysql
        - yarn add cors
        - yarn add react
- In order to run the front end code, run the following command: 'nodemon Index.js'
- Ensure that you are in the correct folder when running the code.

12.8. Mechanical Drawings

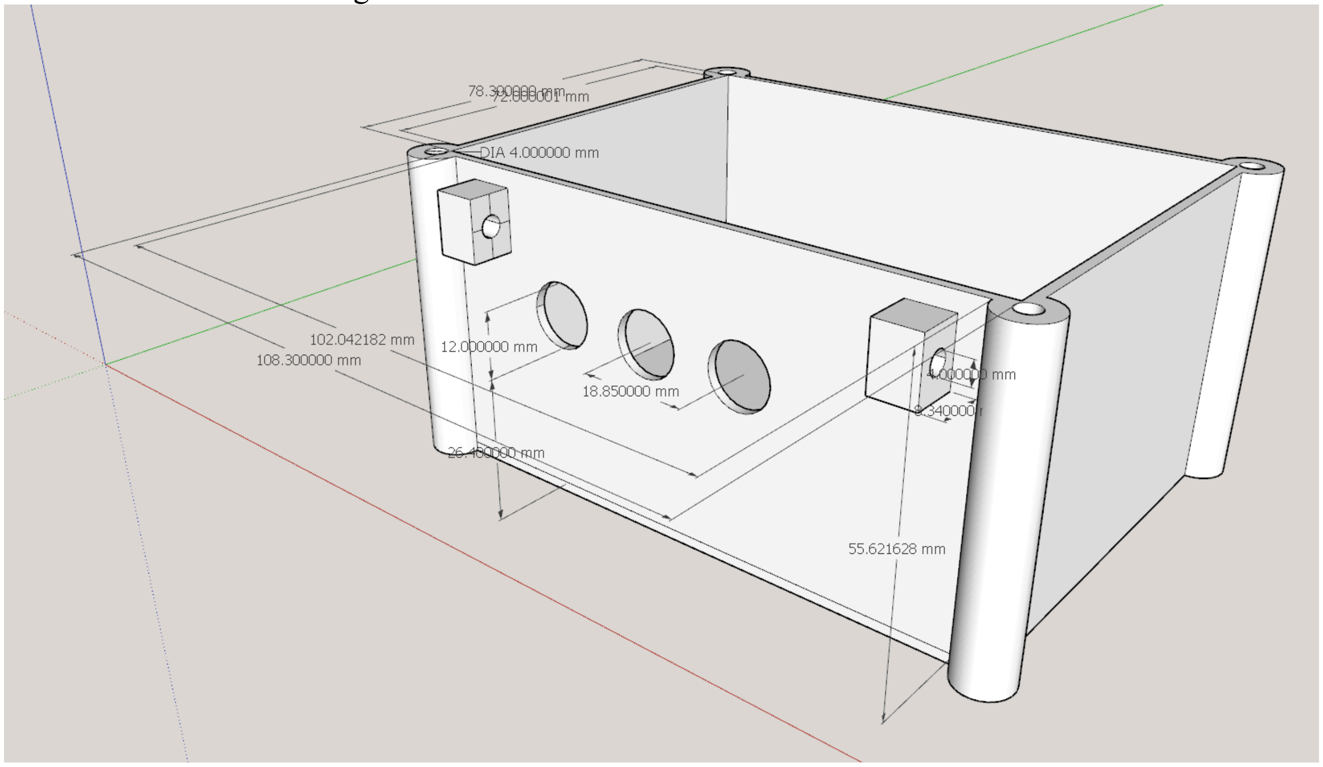


Figure 13 – Enclosure Bottom Piece

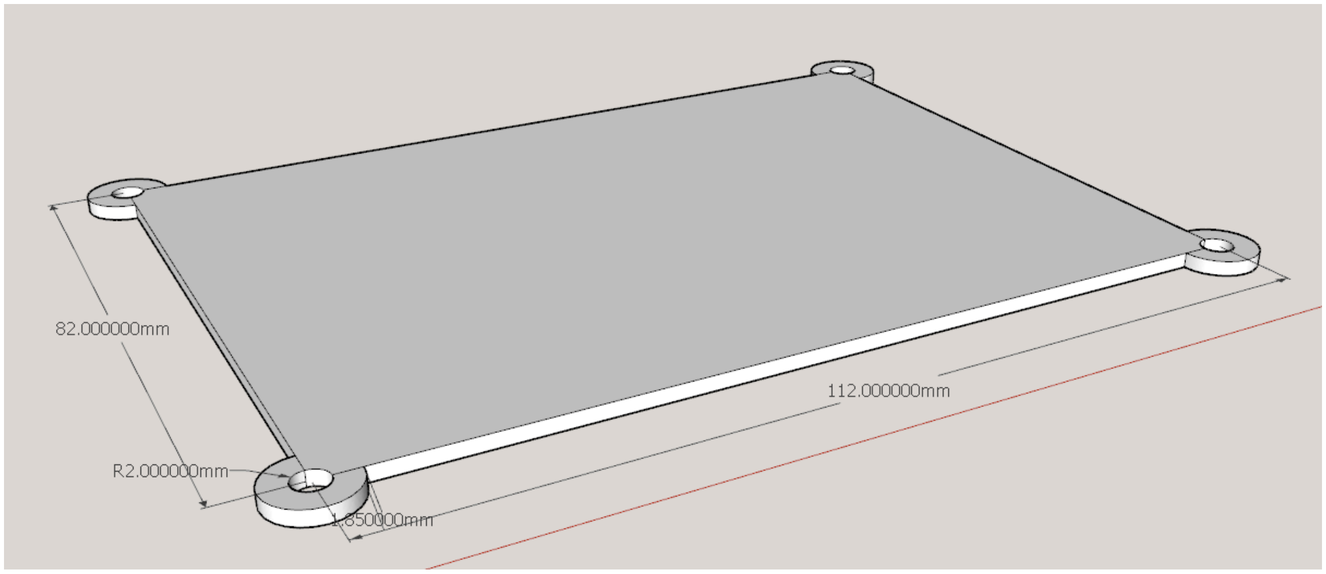


Figure 14 – Enclosure Top Piece

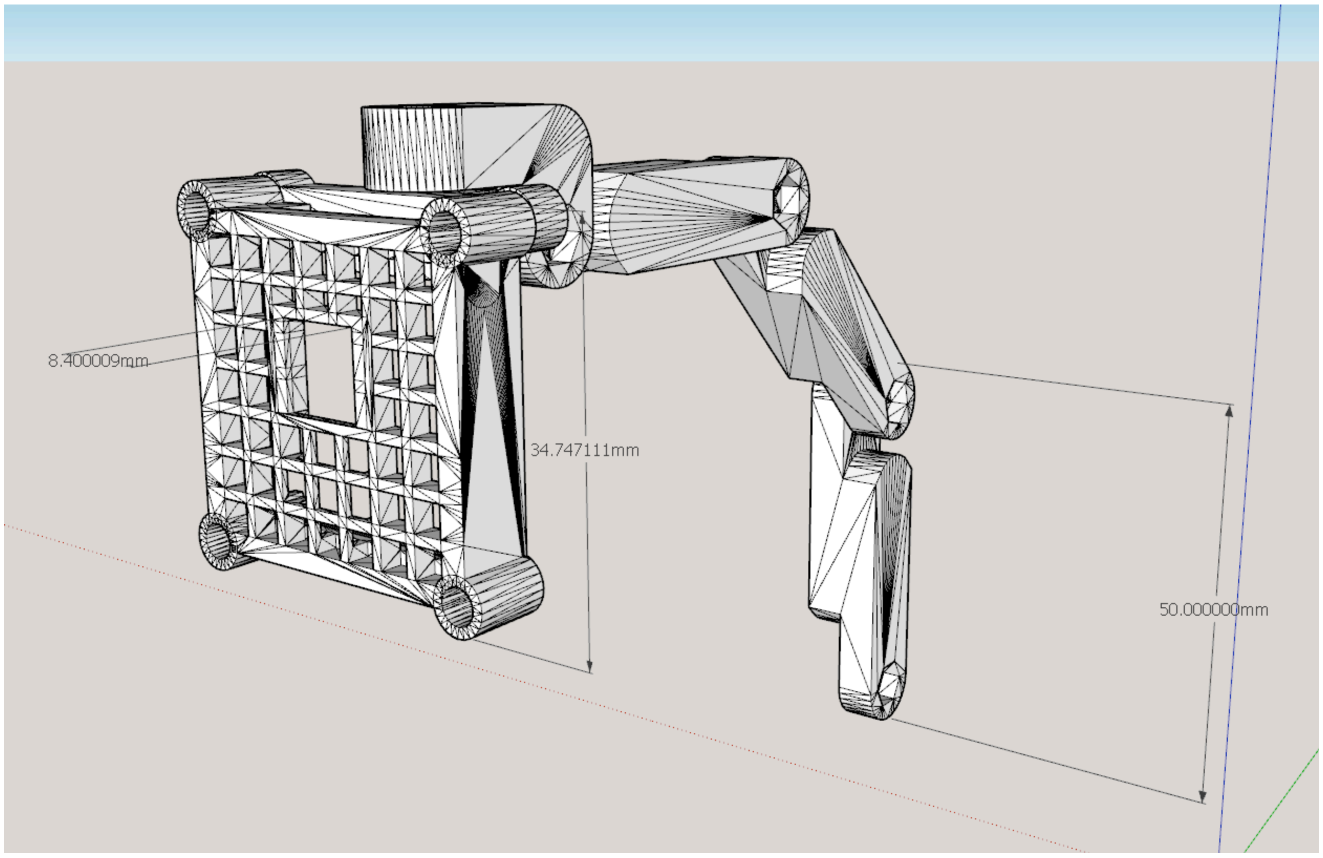


Figure 15 – Raspberry Pi Camera Mount

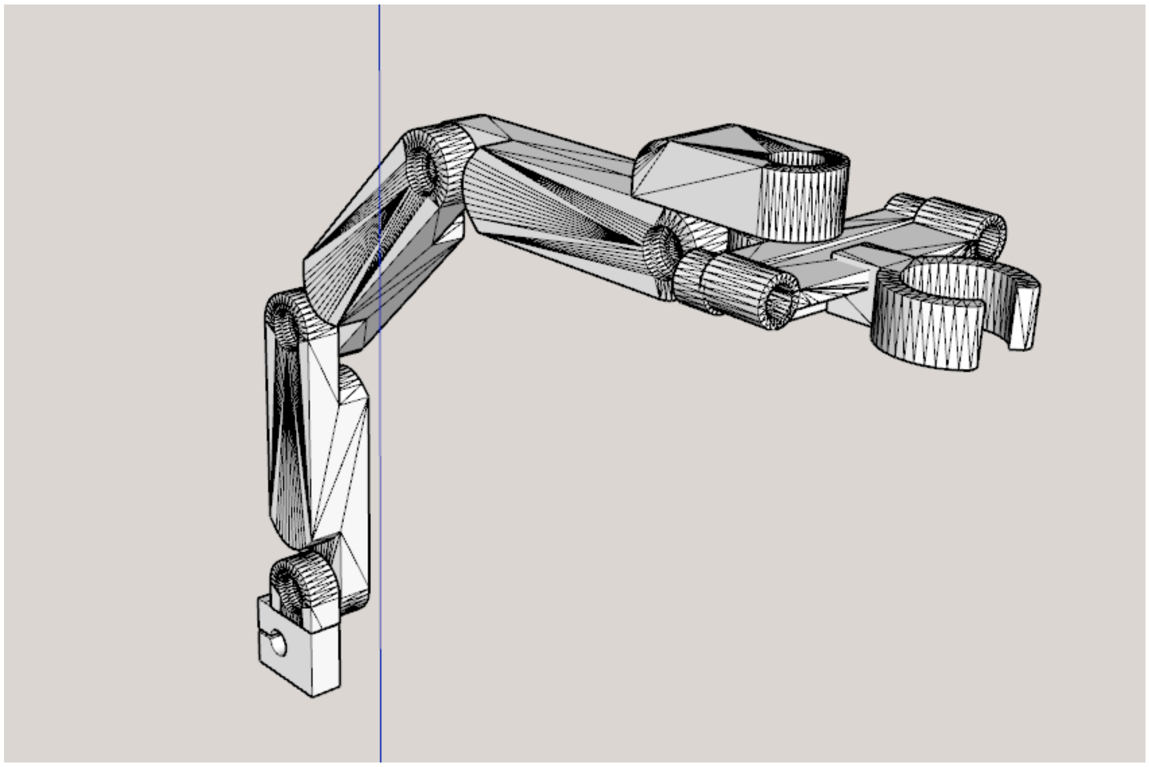


Figure 16 – Sensor Mount

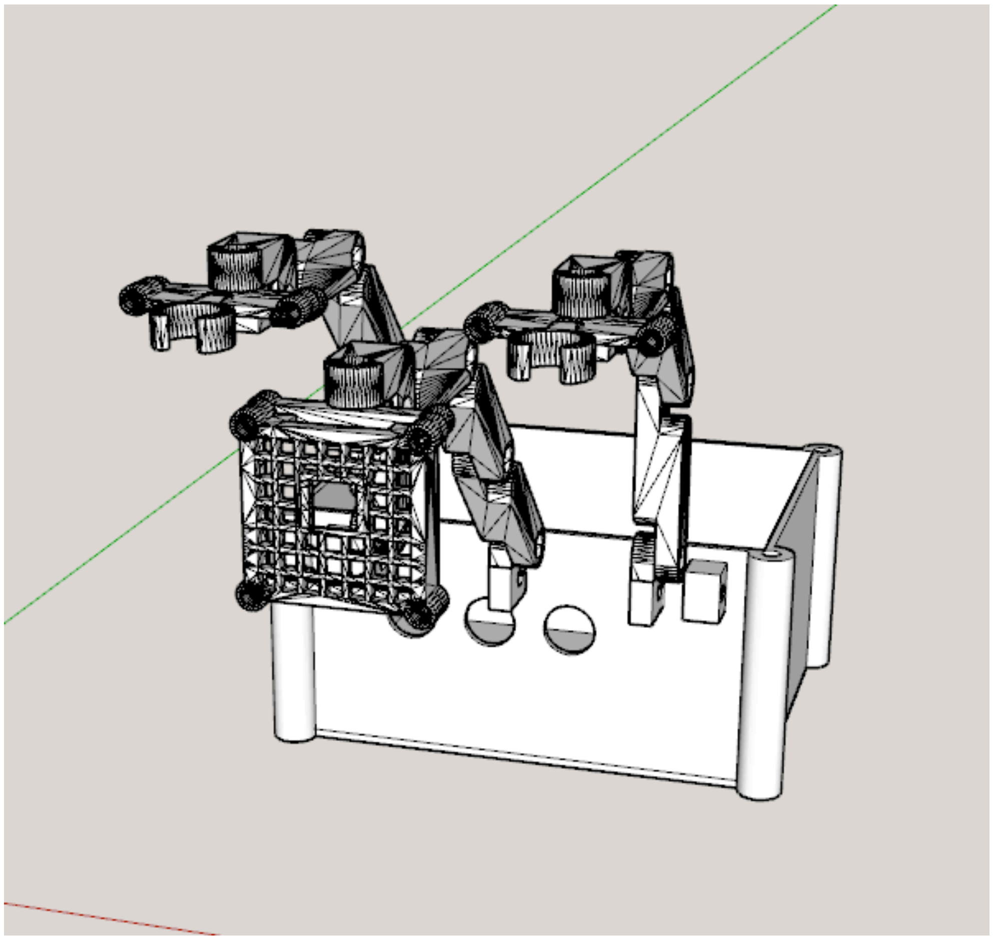
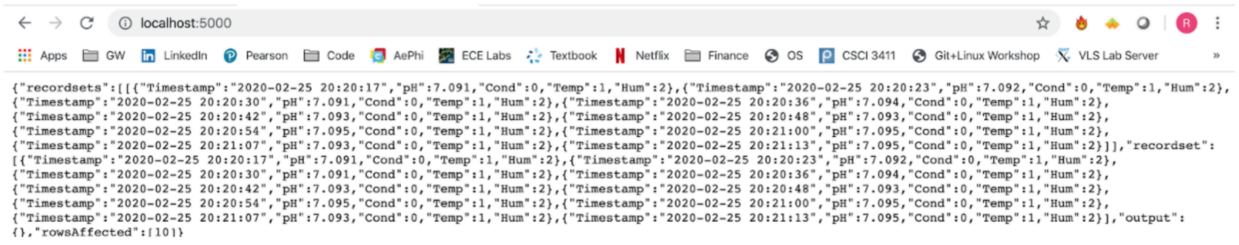


Figure 17 – Overall Enclosure Assembly

12.9. GUI Screenshots



```
{"recordsets":[[{"Timestamp":"2020-02-25 20:20:17","pH":7.091,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:23","pH":7.092,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:30","pH":7.091,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:36","pH":7.094,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:42","pH":7.093,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:48","pH":7.093,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:54","pH":7.095,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:21:00","pH":7.095,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:21:07","pH":7.093,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:21:13","pH":7.095,"Cond":0,"Temp":1,"Hum":2}]],"recordset":
[{"Timestamp":"2020-02-25 20:20:17","pH":7.091,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:23","pH":7.092,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:30","pH":7.091,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:36","pH":7.094,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:42","pH":7.093,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:20:48","pH":7.093,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:20:54","pH":7.095,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:21:00","pH":7.095,"Cond":0,"Temp":1,"Hum":2},
{"Timestamp":"2020-02-25 20:21:07","pH":7.093,"Cond":0,"Temp":1,"Hum":2},{"Timestamp":"2020-02-25 20:21:13","pH":7.095,"Cond":0,"Temp":1,"Hum":2}],"output":
{},"rowsAffected":[10]}
```

Figure 18 – Frontend Output

12.10. Code

```
// written by Rachel Zaltz
function ignoreFavicon(req, res, next) {
  if (req.originalUrl === '/favicon.ico') {
    res.status(204).json({nope: true});
  } else {
    next();
  }
}
var express = require('express');
var cors = require('cors');
var app = express();
var React = require('react');
app.use(ignoreFavicon);
app.use(cors());
app.get('/', function (req, res) {
    var sql = require("mssql");

    // config for your database
    var config = {
                      server:X.X.X.X',
                      user: 'XXXX',
                      port:XXXX,
                      password: 'XXXX',
                      database: 'XXXX'
    };
    // connect to your database
    sql.connect(config, function (err) {

        if (err) console.log(err);
        // create Request object
        var request = new sql.Request();
        // query to the database and get the records
        request.query('select * from capstone.testing.example', function (err, recordset) {
            if (err) console.log(err)
            // send records as a response
            res.send(recordset);

        });
    });
});
```

```
// Listen on a specific host via the HOST environment variable
var host = process.env.HOST || 'localhost';
// Listen on a specific port via the PORT environment variable
var port = process.env.PORT || 5000;
var cors_proxy = require('cors-anywhere');
cors_proxy.createServer({
   originWhitelist: [], // Allow all origins
   requireHeader: ['origin', 'x-requested-with'],
   removeHeaders: ['cookie', 'cookie2']
}).listen(port, host, function() {
   console.log('Running CORS Anywhere on ' + host + ':' + port);
});
var server = app.listen(5000, function () {
   console.log('Server is running..');
});
```

Connection from GUI to Back End Code

```
'''
ECE4925W - Final Capstone Submission, Spring 2020
Phillip Jones, Connor Kraft, Rachel Zaltz

final_atlas - takes data, organizes storage structure, submits queue request for transportation to
storage server
final_atlas_worker - takes resulting data and actually submits to queue
final_atlas_upload - submits data to sql database on storage server

For detailed questions on packages used / scripts, please see User Manual Appendix and the FPR, in
general

'''

import pyodbc                    # Use to make Python connection to SQL Server on storage server
from rq import Queue             # Use to implement Redis queue actions
from rq import get_current_job   # Use to get information on submitted jobs
from redis import Redis          # Use to make Python connection to Redis server on RPi
from time import sleep           # Use to make program stop for period

def upload_to_db(data):
   redis_conn=Redis()
   q=Queue('atlas',connection=redis_conn,default_timeout=300)
   # The following information currently works with the provided server
   # Contact Mark Aglipay if you arrive with issues on the server / with the SQL Server application
   conn = pyodbc.connect(driver='{FreeTDS}',
                 server=X.X.X.X',
                 port='XXXXX',
                 database='XXXX',
                 uid='XXXX',
                 pwd='XXXX')
   cursor=conn.cursor()

   # Following data is taken from the constructed dictionary in final_atlas.py
```

```
    # Data is out of order, but submits to server in correct order, due to how the queueing orders the
information
    # Sleep is used to reduce load on server between sensor readings / data commits
    tval=str(data.values()[3])
    phval=str(data.values()[4])
    condval=str(data.values()[5])
    tempval=str(data.values()[0])
    humval=str(data.values()[2])
    imgval=str(tval + '.png')
    expname=str(data.values()[1])
    cursor.execute('INSERT INTO XXXX.XXXX.XXXX
(Timestamp,pH,Cond,Temp,Hum,img_name,exp_name) VALUES (?,?,?,?,?,?,?)',
tval,phval,condval,tempval,humval,imgval,expname)
    conn.commit()
    sleep(5)
    return
```

Final Atlas Upload Python Code

```
'''
ECE4925W - Final Capstone Submission, Spring 2020
Phillip Jones, Connor Kraft, Rachel Zaltz

final_atlas - takes data, organizes storage structure, submits queue request for transportation to
storage server
final_atlas_worker - takes resulting data and actually submits to queue
final_atlas_upload - submits data to sql database on storage server

For detailed questions on packages used / scripts, please see User Manual Appendix and the FPR, in
general

All information in Class AtlasI2C was taken from Atlas Scientific's sample program for I2C device
communication
'''



from atlas_upload import upload_to_db       # Called script from our application package
import time                                 # Use for actual timing
from time import sleep                      # Use to stop scripts for period of time
import sys                                  # Use to control system level exit of program
import os                                   # Enables user to act on files outside program
import subprocess                           # Use to invoke call to cmd
import picamera                             # Use to control RPi camera
import datetime                             # Use to get current datetime info
import RPi.GPIO as GPIO                     # Use to interact with RPi GPIO pins
import Adafruit_DHT                         # Stock module for temp/humidity sensor
import io                                   # Use to create file streams
from io import open                         # Use to open files in file streams
import fcntl                                # Use to access I2C parameters
import string                               # Use to parse strings
import copy                                    # Use to implement copy as function of Python
from rq import Queue                           # Allows for Redis job queueing
from rq import get_current_job                 # Use for getting information about current job
from redis import Redis                        # Use to make connection to Redis server
import psutil                                  # Use to produce system level diagnostic info about RPi

# Open camera, set resolution
camera = picamera.PiCamera()
camera.resolution = '1080p'

# Open GPIO pin 21, set for output as "flash" for photo
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(21,GPIO.OUT)

# Initialize temp/hum sensor
sensor  = Adafruit_DHT.DHT11
pin = 23

# Initialize queue, open connection to atlas queue
redis_conn = Redis()
q = Queue('atlas', connection=redis_conn, default_timeout=300)

'''
All information in AtlasI2C is taken from the provided configuration of Atlas' I2C devices

This information can be altered as desired, please tread lightly when doing this,
as this can result in significant differences in operation for these sensors

In addition, this version has been altered, as the provided build encountered problems
with sensor configuration and operation
'''

class AtlasI2C:
    long_timeout = 1.5                         # Timeout needed to query readings / calibrations
    short_timeout = 0.3                        # Timeout for regular commands
    default_bus = 1                            # Default RPi I2C bus
    default_address = 98                       # Default sensor address
    LONG_TIMEOUT_COMMANDS = ("R", "CAL")    # Wait for long period while reading
    SLEEP_COMMANDS = ("SLEEP", )               # Assign sleep commands

    def __init__(self, address = default_address, bus = default_bus):
        # Init 2 file streams, for reading and writing
        self._address = address or self.default_address
        self.bus = bus or self.default_bus
        self._long_timeout = self.long_timeout
        self._short_timeout = self.short_timeout
        self.file_read = io.open(file="/dev/i2c-{}".format(self.bus), mode="rb", buffering=0)
        self.file_write = io.open(file="/dev/i2c-{}".format(self.bus), mode="wb", buffering=0)
        self.name = ""
```

```
def set_i2c_address(self, addr):
    # Set I2C communications to slave address
    I2C_SLAVE = 0x703
    fcntl.ioctl(self.file_read, I2C_SLAVE, addr)
    fcntl.ioctl(self.file_write, I2C_SLAVE, addr)
    self.address = addr

def write(self, cmd):
    # Appends null character and send string over I2C
    cmd += "\00"
    self.file_write.write(cmd.encode('latin-1'))

def handle_raspi_glitch(self, response):
    if self.app_using_python_two():
        return list(map(lambda x: chr(ord(x) & ~0x80), list(response)))
    else:
        return list(map(lambda x: chr(x & ~0x80), list(response[1:])))

def app_using_python_two(self):
    return sys.version_info[0] < 3

def get_response(self, raw_data):
    if self.app_using_python_two():
        response = [i for i in raw_data if i != '\x00' ]
    else:
        response = raw_data

    return response

def response_valid(self, response):
    valid = True
    error_code = None
    if(len(response)>0):
        if self.app_using_python_two():
            error_code = str(ord(response[0]))
        else:
            error_code = str(response[0])

        if error_code != '1':
            valid = False

    return valid, error_code

def get_device_info(self):
    if(self.name == ""):
        return str(self.address)
    else:
        return str(self.address) + " " + self.name

def read(self, num_of_bytes=31):
    raw_data = self.file_read.read(num_of_bytes)
```

```
        response = self.get_response(raw_data=raw_data)
        is_valid, error_code = self.response_valid(response=response)

        if is_valid:
            char_list = self.handle_raspi_glitch(response[1:])
            result = "Success " + self.get_device_info() + ": " + str(''.join(char_list))
        else:
            result = "Error " + self.get_device_info() + ": " + error_code

        return result

    def get_command_timeout(self, command):
        timeout = None
        if command.upper().startswith(self.LONG_TIMEOUT_COMMANDS):
            timeout = self._long_timeout
        elif not command.upper().startswith(self.SLEEP_COMMANDS):
            timeout = self.short_timeout

        return timeout

    def query(self, command):
        self.write(command)
        current_timeout = self.get_command_timeout(command=command)
        if not current_timeout:
            return "sleep mode"
        else:
            time.sleep(current_timeout)
            return self.read()

    def close(self):
        self.file_read.close()
        self.file_write.close()

    def list_i2c_devices(self):
        prev_addr = copy.deepcopy(self._address)
        i2c_devices = []
        for i in range(0,128):
            try:
                self.set_i2c_address(i)
                self.read(i)
                i2c_devices.append(i)
            except IOError:
                pass
        self.set_i2c_address(prev_addr)

        return i2c_devices

def main():
    # Allow for user input of experiment name
    # MUST BE IN QUOTES, NO SPACES
    experiment_name = input("Input the name of the experiment: ")
    # Timestamp name of data log file and enable write
    d2 = datetime.datetime.now()
    # Create local data dump folder
    os.system('mkdir /home/pi/Desktop/capstone/' + experiment_name + '.' +
d2.strftime('%Y.%m.%d-%H.%M.%S'))
    # Create local data dump csv
    csv = open('/home/pi/Desktop/capstone/'+ experiment_name + '.'+ d2.strftime('%Y.%m.%d-
%H.%M.%S') + '/' + d2.strftime('%Y.%m.%d-%H.%M.%S') +'.csv', 'a')
    # Create I2C port object
    device = AtlasI2C()
    try:
        # FOR LOOP HARD CODED, CHANGE IN FUTURE (NUMBER OF READS)
        # This loop turns on/off flash, takes photo in between, reads from each sensor
        # Sensor info and system diagnostics are then organized and submitted to the Redis queue for
storage server
        # All info is then also saved to a local CSV, to be also transferred to storage server via rsync
        for i in range(0,5):
            d = datetime.datetime.now()
            print('\n')
            print(d)
            pict = d.strftime('%Y.%m.%d-%H.%M.%S')
            picn = ('/home/pi/Desktop/capstone/' + experiment_name + '.'+ d2.strftime('%Y.%m.%d-
%H.%M.%S') + '/' + pict + '.png')
            print('Light on\t')
            GPIO.output(21,GPIO.HIGH)
            sleep(0.5)
            print('Taking picture\t')
            camera.capture(picn)
            print('Light off, taking sensor data')
            GPIO.output(21,GPIO.LOW)
            #humidity, temperature = Adafruit_DHT.read_retry(sensor,pin)
            # CHANGE THESE TEMP/HUM HARD CODED VALUES IN FUTURE
            temperature = 1.00
            humidity = 2.00
            device.set_i2c_address(99)
            phstring = device.query('R')[11:]
            phstring = str(phstring)
            device.set_i2c_address(100)
            condstring = device.query('R')[12:]
            condstring = str(condstring)

            data = {
                'time' : d.strftime('%Y-%m-%d %H:%M:%S'),
                'pH': phstring,
                'conductivity': condstring,
                'temperature': temperature,
                'humidity': humidity,
                'exp_name': experiment_name
                }
            log_data = {
                'CPU usage' : psutil.cpu_percent(),
                'CPU temp' : subprocess.check_output(['vcgencmd', 'measure_temp'],
stderr=subprocess.STDOUT),
                'RAM usage' : psutil.virtual_memory()[2]
                }
            job = q.enqueue(upload_to_db, data, result_ttl=86400)
            print('Current job on queue: %s\n' % (job.id,))
            # Parsing and handling this information is both annoying and difficult,
            # I reccomend leaving this scheme for the entirety of the project
            # i.e. I reccomend not moving away from CSV as a secondary storage method
            csv.write(unicode('experiment: ' + str(data['exp_name']) + ','))
            csv.write(unicode('time:  ' + str(data['time']) + ','))
            csv.write(unicode('pH:  ' + str(data['pH']) + ','))
            csv.write(unicode('conductivity:  ' + str(data['conductivity']) + ','))
            csv.write(unicode('temperaure:  ' + str(data['temperature']) + ','))
            csv.write(unicode('humidity:  ' + str(data['humidity']) + ',\n'))
            csv.write(unicode('CPU usage:  ' + str(log_data['CPU usage']) + ','))
            csv.write(unicode('CPU ' + str(log_data['CPU temp'])[:-1] + ','))
            csv.write(unicode('RAM usage:  ' + str(log_data['RAM usage']) + ','))
            csv.write(unicode('\n\n'))
            sleep(3)
            continue

        # Proper exit: close camera, file, and invoke cmd to employ SSH file sync
        # Then exit at a system level
        camera.stop_preview()
        csv.close()
        subprocess.call(['rsync','-avrPzh','/home/pi/Desktop/capstone','-e','ssh -p
22','user@X.X.X.X:~/'])
        sys.exit()

    # Improper exit: close camera, file, and brute force exit system
    except KeyboardInterrupt:
        camera.stop_preview()
        csv.close()
        print('\nClosed by keyboard interrupt.')
        sys.exit()

if __name__ == '__main__':
    main()
```

Final Atlas Python Code

```
'''
ECE4925W - Final Capstone Submission, Spring 2020
Phillip Jones, Connor Kraft, Rachel Zaltz

final_atlas - takes data, organizes storage structure, submits queue request for transportation to
storage server
final_atlas_worker - takes resulting data and actually submits to queue
final_atlas_upload - submits data to sql database on storage server

For detailed questions on packages used / scripts, please see User Manual Appendix and the FPR, in
general

This program was provided entirely via the RQ setup guide

Launch Redis server first, type the following on cmd: "redis-server"

Launch with the following on cmd: "python final_atlas_worker.py atlas"

This names the queue that jobs will be submitted to

This script acts as middleman transport, can be left running at all times (script works automatically)
'''

import sys
from rq import Connection, Worker

with Connection():
    qs = sys.argv[1:] or ['default']
    w = Worker(qs)
    w.work()
```

Final Atlas Worker Python Code