Final Report: XPRIZE Ocean Sensor

Pritak Patel, Ryan Anders, Sam Kelly

26 April 2017

Table of Contents

I. Introduction

Task

For Spring 2017 ECE 449, Sensors and Sensor Interface Design, each team is tasked with the design, manufacture, and operation of an ocean sensing system. This system is intended to address the sensing section of the Ocean XPRIZE challenge; the final product will contain a colorimetric sensor for Rhodamine WT relative dye concentration and an electrochemical pH sensor, as well as electronic interfaces for both sensors [1]. For receiving commands and the transmission of data, the system will employ wireless communication.

Design Specifications

The system's microcontroller will be the Adafruit Feather HUZZAH (with built-in ESP8266 Wi-Fi chip) and the entire system will be operated by a 3.7 V Lithium Ion battery [2]. The colorimeter dye tracer system must be able to rank different samples of ocean water by relative concentration of Rhodamine WT, which will range from 0-40 ppm (and differ by at least 5 ppm) with sample turbidity ranging from 0.2-10 NTUs. For the final demo, three different non-turbid samples were tested, due to the discovery and confirmation by all groups that this planned range of sample turbidity did not have an effect on the groups' colorimeter systems. The electrochemical pH sensor must be capable of identifying a sample's pH, within the range from 3-7, with 0.2 accuracy. The Wi-Fi communication system must allow for a computer to send a wireless signal to initiate testing, and for gathered data to be sent wirelessly back to the computer, which will display the pH and colorimeter data. The system must be IEEE Wi-Fi Standard-compliant. Because the system will be dunked in ocean water the package must be watertight, and no human intervention will be allowed between initiating testing and receiving data. Prototyping cost, environmental impact, power consumption, and battery life will all be reported.

II. Project Management

To facilitate a workflow that is both productive and efficient, the various segments of the project were divided up among group members. While all members are still knowledgeable about the entirety of the project, placing members "in charge" of each component enabled work on those

components to remain focused while still adequately allowing for man-hours to be spread across the project. The colorimetry skill-set development mini project is the only component that was not delegated to a specific group member. The other assignments are as follows:

- Pritak - Wireless Communication, pH Sensing, Ordering and Cataloguing Parts
- Ryan - Turbidity Consideration/Calibration, pH Sensing, General Circuitry/System Power
- Sam - 3D Printing, Waterproof Housing

A Gantt chart was also created to ensure that time was allocated appropriately to the various sections of the project. A free trial of online software at teamgantt.com was used to create and consult the Gantt chart reproduced in Figure A1 of the Appendix.

III. Colorimetry

Background

Rhodamine WT dye is a bright fluorescent red dye that is widely used for ocean water tracing applications. Because Rhodamine WT dye causes the color of a sample to change, the relative concentration of dye can be determined by the color of the sample--solutions with higher concentrations of dye will have a more intense red or pink color. Therefore, throughput data from a colorimeter should naturally map to relative concentrations of dye.

A basic colorimeter requires only a light source pointed at a sample, and a photodetector on the opposite side of the sample directly across the photodetector. Rhodamine WT is a fluorophore--when excited with green light (peak excitation at 558nm), the dye emits light of its own (peak emission at 582nm) [3]. Hoping to use this phenomenon to trigger additional changes in sample color outside of just dye concentration, a green LED was chosen initially to be the light source. As will be explained later, our final design no longer worked with green light once we switched to using a vinyl tube instead of a cuvette as the surface for readings. To get accurate readings from our colorimetry system we resorted to using a white LED.

In the original design requirements, relative concentrations of Rhodamine WT needed to be determined in samples with varying turbidity. Turbidity is the cloudiness or haziness of a fluid caused by large numbers of suspended particles (in our case silica) [4]. In terms of LED throughput to the color sensor, turbidity in theory should disperse the light through scattering and

affect accurate readings. To solve this problem we originally planned to use a color sensor placed at 90 degrees from the throughput sensor to collect scattered light. By comparing throughput readings with the 90 degree sensor readings, an accurate colorimeter system could be calibrated to account for turbidity. As will be explained later however, the turbidity in the solutions did not effect the throughput readings enough to warrant using the data from the additional sensor in our final design.

Design Goals

Design specifications require the relative identification of Rhodamine WT dye with resolution 5 ppm given solutions between 1 ppm and 40 ppm at varying levels of turbidity (0.2-10 NTUs). A fully automated system must facilitate the collection of data and wireless transmission to a computer. The system must also be watertight to prevent damage to the electronic components.

Adafruit NeoPixel RGBW LED

The Adafruit NeoPixel RGBW LED is a small (5x5mm) LED with an integrated driver circuit. This LED was chosen as it has an associated Arduino library to control LED intensity, duration and color [5]. For our skill-set mini development system, green light at 558nm was our chosen light source (see: background). In final iterations of our project, the LED needed to be switched to white light as will be discussed later.

Adafruit TCS34725 Color Sensor

Two Adafruit TCS34725 Color Sensors were chosen for the throughput and 90-degree turbidity sensors due to their built in Transimpedance Amplifiers (TIA) and Arduino library [6]. The sensors have onboard white LEDs which are unused. Each sensor outputs red, green, blue, lux, and color temperature data readings, which was useful in initial testing to determine which variables to track. Additionally, the inbuilt 3.3V voltage regulator allows for power from 3-5V DC. One drawback to the TCS34725 is that it has a fixed I2C address that can not be changed. When using two of the sensors, this made it impossible to successfully get readings from either sensor. To solve this problem, the power pins of the two sensors were attached to digital logic pins on the Feather Huzzah, and readings could be obtained by reading from one sensor at a time, using the logic pins to switch on the desired sensor and switch off the other sensor. Unfortunately, this process caused the first three readings to be corrupted every time a sensor

was switched on. Waiting to collect data until these three readings have passed, as well as giving each sensor adequate time to turn on, caused data collection to take significantly longer. However, overall the TCS34725 has proven easy to use and fully sufficient for the design requirements.

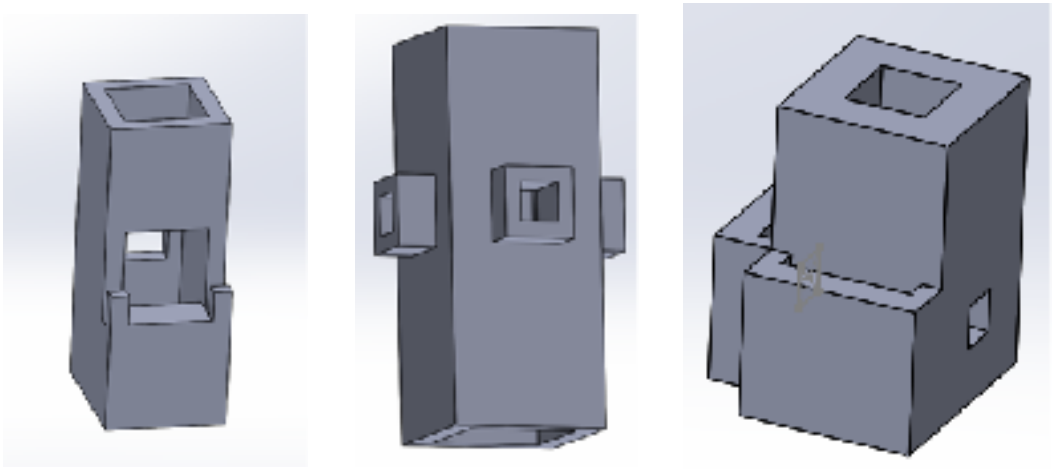


Figure 1. (a) Initial colorimetry design, (b) second iteration of design, (c) final testing design including slide-in holes for color sensors.

Design Evolution

Over the course of the semester our colorimetry system changed design the most of any of the other main components. Our initial design, shown in Figure 1a, only had a cut-out for one throughput sensor and one LED, as well as not taking into account light leakage in any way. After getting good, but unreliable results in our skill-set development mini project we looked to light proof the system immediately. In our second iteration of design we created an outside shell for the colorimeter as well as added a turbidity sensor hole, however after finding out in weekly presentations that 3D printed materials were not waterproof, we reverted our system to a more basic setup with 3 holes for the sensors and LED (Figure 1b). Not realizing the difficulties of keeping the electronic components in place, for our final testing colorimeter setup we created slide-in holes for the two sensors and a hole that matched the LED in size so that it could be hot-glued into place (Figure 1c). To prevent light leakage in this testing apparatus we placed all the electronics and colorimeter in a cardboard box and shut it before starting tests.

Final Design

After enough tests were run on the testing apparatus and the final overall housing was ready, we adapted the 3D printed colorimeter model to slide into our inside casing easily as shown in Figure 2a. The full system was tested a few times with a cuvette in the colorimetry system

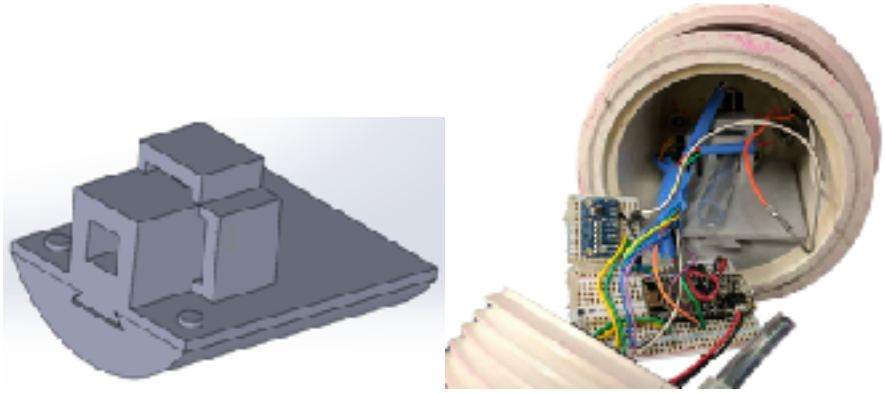


Figure 2. (a) 3D CAD model of final slide-in colorimeter, (b) Fully assembled colorimeter system from final demonstration

connected on both ends to the pipes that ran through the middle and out of the housing on both ends. However, we found that keeping the cuvette joints completely waterproof while also allowing the system to be completely opened and disassembled easily was very challenging. Instead, the cuvette was swapped out for the same vinyl tube that ran through the middle of the housing to create a fully waterproof system (Figure 2b). To ensure this final system with the vinyl tube was reliable and produced repeatable results, we performed a few tests which are discussed in detail in the next section. In completing these tests we also found that green light no longer worked as it did in the skill-set development mini project. High concentrations of Rhodamine WT dye made the throughput sensor read high levels of red light but very low levels of other light - which was the trend we sought out for but was not repeatable at lower concentrations. After running a few tests discussed later, we found using white light and only reading lux values from the sensor gave us the most reliable and repeatable results. Lux was calculated by our Adafruit TCS34725 sensors using the formula: (-0.32466 * red) + (1.57837 * green) + (-0.73191 * blue), where red, green, and blue were the values read from the sensor for each of the colors [6]. It is important to note that the turbidity sensor was placed in our final design but the data produced from it was not used in any form.

Turbidity Testing Process

To assess the effects of turbidity on readings from the two TCS34725 sensors, a testing environment that closely resembled the previous setup for the skill-set development mini project was created. Cuvettes were filled with the provided turbid samples (1, 3, 5, 7, and 10 NTUs), placed into the 3D-printed housing, and closed inside of a light-proof box. Arduino and MATLAB scripts worked together to take 10 readings per sensor, one every second, and plot the resulting data (scripts discussed further in Testing Process section below).

Turbidity Results

Although turbidity was in our original design requirements, after completing the turbidity tests, we found that the values the color sensor was reporting were essentially noise: the differences in values between samples were extremely small (negligible in comparison to throughput readings) and did not appear to have any discernible relationship to sample turbidity. The plots of this data can be seen in Figure 3--lux plots are shown because lux is calculated from the red, green, and blue values, and thus is a concise but complete way to identify trends in the color sensors' readings. We hypothesize that this was due to the small concentrations of turbidity used in this project and thus kept the turbidity sensor in our final design in hopes that the components would already be in place for the system to work in ocean water that was more turbid as a possible extension of our project in the future.

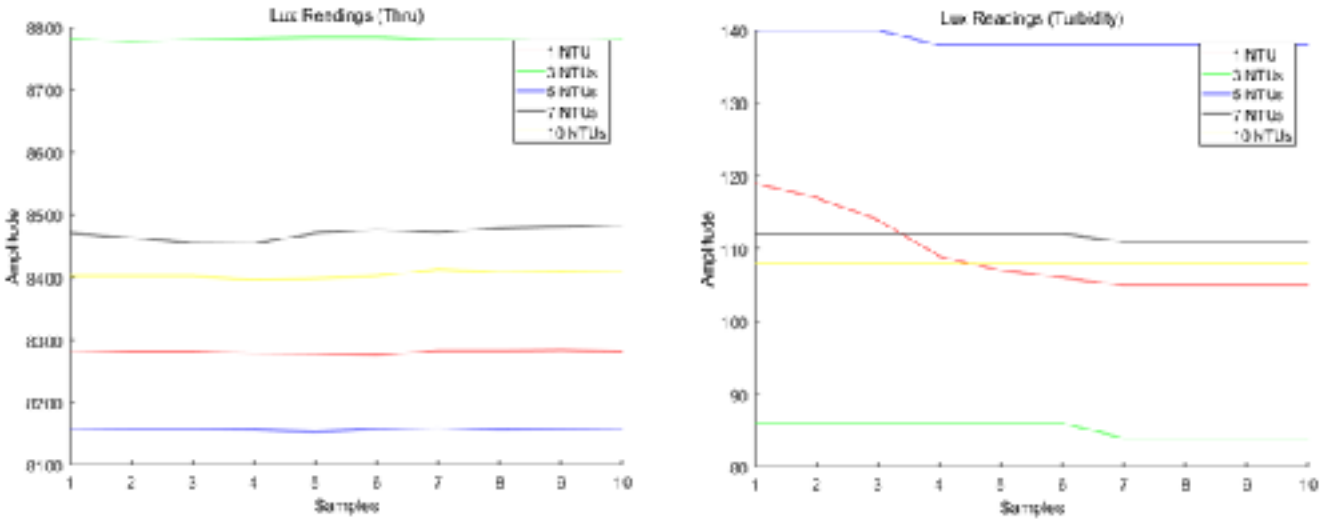


Figure 3. Lux values of throughput color sensor (Left) and 90-degree turbidity color sensor (Right) for samples with varying turbidity.

Testing Process

The design of our overall project as well as the colorimetry subsystem allowed testing the system to be a quick and simple process. After sending a "start" signal wirelessly, the system was submerged vertically. Because the tubing travels relatively straight through the final product from the top to the bottom, vertically dunking the system into samples allowed water to very easily enter the sample chamber for data collection and exit upon the system's removal from the water, all without any bubbles forming. Due to memory constraints pertaining to the wireless communication system, only three lux readings were taken from each of the two sensors after the system was submerged. However, standard deviations were found to be extremely low across all trials, so the small amount of readings had little effect on results. After data collection was completed and the system was removed from water, the data was sent wirelessly to ThingSpeak. A MATLAB script run on a computer used the ThingSpeak API to retrieve the values, plot the retrieved data, and order the tested samples from lowest to highest concentration of Rhodamine WT using averages of the three lux readings taken on each sample.

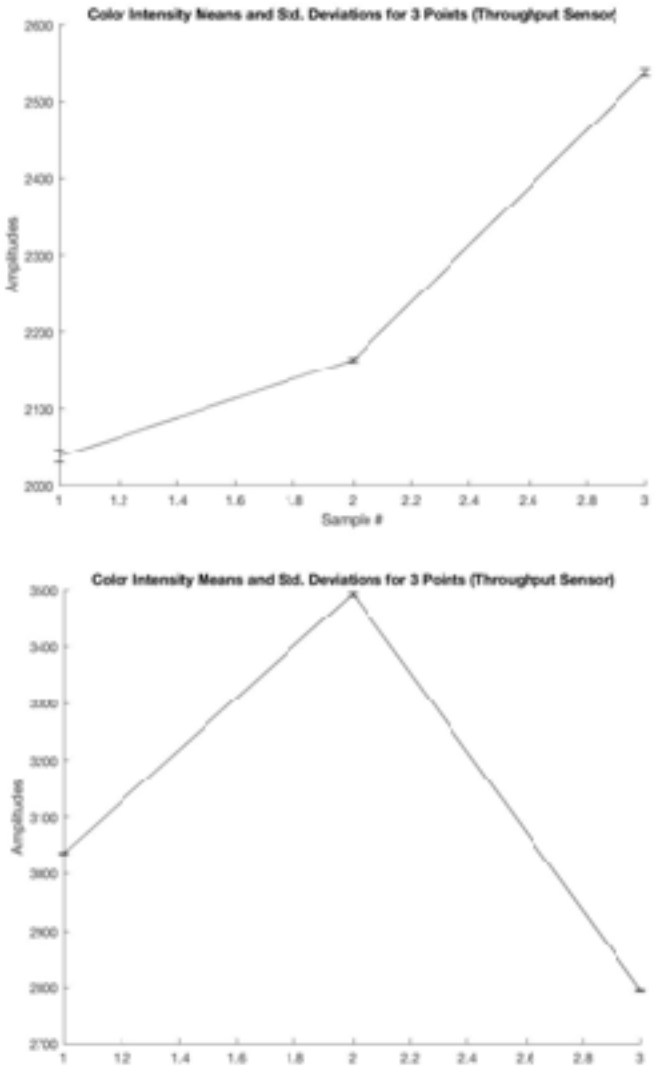


Figure 4. Lux readings from 3 samples of differing Rhodamine WT concentrations. Samples were tested in different order between tests.

Results

Due to a late switch to a tube-only sample chamber caused by problems with waterproofing interfaces between the tubing and the cuvette, testing time for the final design of the colorimeter subsystem was very limited. However, we were able to successfully perform and document two tests in perfectly-replicated demo day situations using three

samples containing concentrations of Rhodamine WT that differed with about 5 ppm. In both tests, standard deviations were extremely low and the samples were correctly ordered both times. Despite values being inconsistent when compared across the two tests, it can be clearly seen in Figure 4 that although the samples were somewhat close in Rhodamine WT concentration, the colorimeter had no issues differentiating between the three samples. We hypothesize that the variation across tests was caused by differing light leakage situations--both a clear bucket and an orange, opaque bucket were used for testing. Colorimeter values were also monitored during the extensive pH testing that followed, and identical behavior and results were seen every time the system was run. The data obtained from this colorimeter, using the clear tube to hold the sample rather than a cuvette, appears to be more precise and have finer resolution than the data gathered with a system that used the cuvette (Figure 5). We believe that this greater resolution was caused by the cylindrical tube acting as a lens that is able to focus the light from the LED onto the throughput color sensor. This prevents excess light from bouncing around the inside of the colorimeter housing and could increase the accuracy of readings.

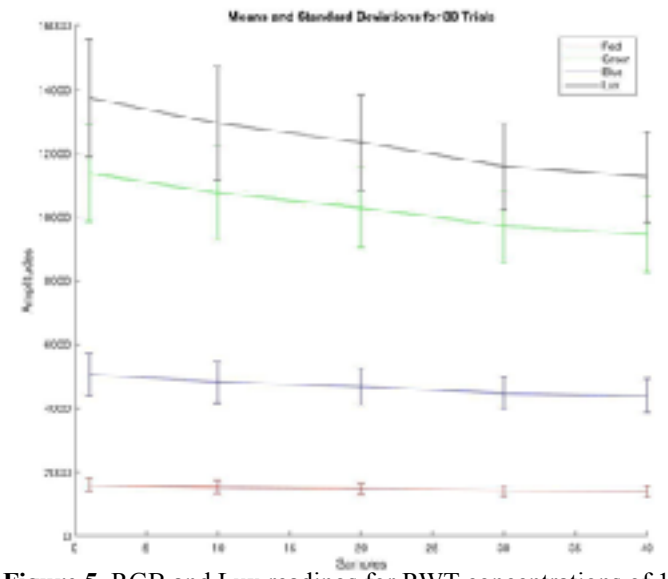


Figure 5. RGB and Lux readings for RWT concentrations of 1, 10, 20, 30, and 40 ppm. Obtained using cuvette-based colorimeter.

IV. Electrochemical pH Sensing

Background

Aside from glass electrodes, which are commonly used to measure pH commercially, other forms of electrochemical pH sensing are relatively poorly documented. Given four metal rods: Zinc, Aluminum, Stainless Steel, and Titanium, designing a robust system that could accurately and precisely measure pH required extensive background research.

Bimetal corrosion or galvanic corrosion is a process by which current is created using two different metals as a cathode and an anode when placed in water. The driving force of the

current is the difference in electric potential between the two metals. The metal that acts as the anode is always the electrode that has the lower potential, and vice versa for the cathode [7]. Using this information, we found galvanic series charts (Figure 6) that helped us decide which metals to use in our design. Zinc has the lowest potential of all the metals provided, hence it is used as ground in our design. Since Aluminum has the closest electric potential to Zinc, we hypothesized that we would see the least varying voltage difference between those two and decided to forgo using the Aluminum rod.

Metal	Steady State Electrode Potential, Volts
Titanium	-0.1
Type 316 Stainless Steel	-0.05
Type 304 Stainless Steel	-0.08
Type 410 Stainless Steel	-0.15
Aluminum 3003-H	-0.79
Zinc	-1.03

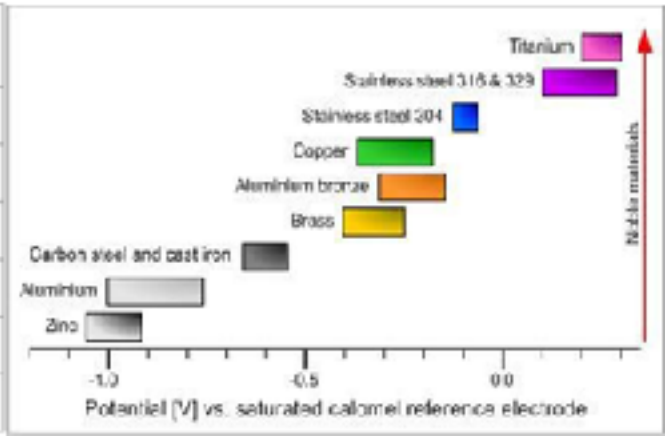


Figure 6. Galvanic series in flowing sea water on the left [7]. Galvanic series versus reference calomel electrode on right [8].

In a previous study done by Brooke et al., Zinc was also used as ground, and Titanium and Stainless Steel, among others, were used to measure electric potential differences and map them to pH [9]. The study found that, as we hypothesized, the electric potentials between the metals were accurate enough to measure pH with specific temperatures and timing. However, there were some notable differences in experimental setup. Brooke et. al., studied the electrochemistry of the metals over a longer period, had larger rods, and experimented in actual ocean water. As we developed our design we attempted to use previous studies as a foundation but also keep in mind how time, temperature, and different metal rod sizes may affect our results.

Design Goals

Given Zinc, Aluminum, Stainless Steel, and Titanium rods, an electrochemical pH sensing unit must be created that will be able to ascertain the pH of a sample within 0.2 of the sample's pH level, within a range of 3-7. An electronic interface must also be created to facilitate the collection of data from the metal rods and its delivery to the microcontroller. Considering the

presence of these electrical components, the unit will need to be watertight. Considerations must be made in the system design and data collection process to attempt to compensate for the effects of drift in voltage readings over time. As always, the ultimate goal is to design a sensing system that is robust and accurate, but also efficient in both power consumption and overall cost.

16-bit ADC

At the forefront of the electronic interface between the metal rods and the microcontroller is the Adafruit ADS1115 16-Bit Analog-to-Digital Converter (ADC). The precision, versatility, and simplicity of this component make it an easy choice for this section of the project. This 16-bit ADC allows for the readable voltage range to be split into 65,536 discrete values, resulting in much more precision than its 12-bit counterpart (4096 discrete values) [10]. The component uses I2C 7-bit addresses between 0x48-0x4B, which can be selected using jumpers. This allows for easy integration into the finished product, as the ADCs I2C address will not conflict with any other components used. The ADC also comes with an easy-to-use and well-documented Arduino library, allowing for easy coordination with the Feather HUZZAH microcontroller.

Metal Rods

Four different metal rods were provided to the group: Zinc, Aluminum, Stainless Steel, and Titanium. All of the rods are ¼" in diameter and 3" in length, except for the Zinc rod, which has the same diameter but is 2" in length. As long as the amount of metal submerged in water is kept consistent across the rods and proper calibrations are performed, this difference in rod length should not affect results. Considering the values in Figure 3, we chose Zinc to serve as the anode and Titanium to serve as the cathode for preliminary testing, then embedded these same two rods into the final system.

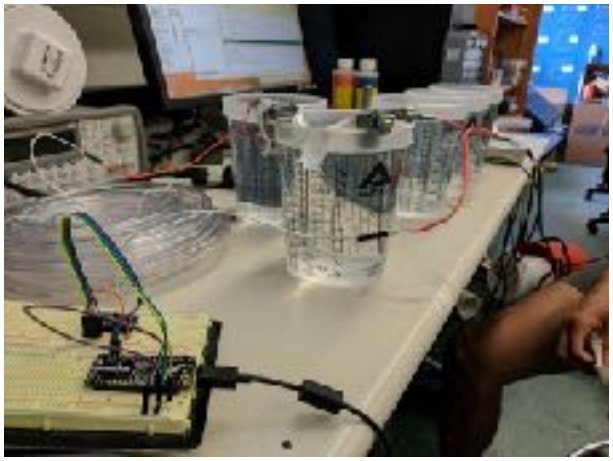


Figure 7. Preliminary pH testing environment setup.

Preliminary Testing Setup

For preliminary pH testing, the Zinc and Aluminum rods were placed into metal saddle connector clamps and secured onto the sides of 32 oz mixing cups containing the pH solutions (Figure 7). Each of the 32 oz mixing cups were carefully filled to the same height to attempt to provide consistency between the different solutions. Due to the rods being different lengths, the Titanium rod was clamped 1" higher than the Zinc rod, ensuring that both rods had the same amount of surface area submerged. Wires were attached to the metal clamps via alligator clips and connected to the A0 (Titanium) and A1 (Zinc) ports on the 16-bit ADC. The ADC is then connected to the SDA/SCL I2C ports on the Feather Huzzah, which in turn was, for the duration of preliminary testing, connected directly to a desktop via USB. This connection allowed us to bypass the wireless communication system and record a large quantity of data points in a fairly short period of time.

Figure 8. Metal rods, rings, and wires potted in epoxy.

Final Design

After completing preliminary pH testing, the sensor needed to be added to the final product. Because the major components of the sensing system consisted of only the two rods and the ADC, this transition was relatively straightforward. Two holes were drilled in the bottom of the PVC casing. The rods were inserted into the holes with slightly over 1" of each rod left exposed to the outside of the system. Small metal rings with wires soldered to their sides replaced the saddle clamps and alligator clips, and the rod, rings, and wires were potted in about 1/2" of epoxy to secure everything in place (Figure 8). The interface between PVC and the metal rods on the system's exterior was treated with Loctite marine sealant to prevent water from leaking into the system. The ADC and electrical connections were soldered to a PCB and mounted, along with the Feather Huzzah, on a 3D-printed sliding plate using hot glue.

Because the rods were now secured in place, this iteration of the pH sensor provided a very consistent environment for data collection. Also, having the entirety of the pH-sensing

system contained in the lower half of the final product allowed us to dunk only that half of the system into solutions and almost perfectly replicate the testing environment of the final demo, even before any of the other systems were completed or the full product was completely watertight.

Testing Process

Before testing each solution the rods were dried and sanded by hand with 400-grit sand paper, and an electronic pH meter was used to obtain a last-second pH reading for data analysis. After the rods were submerged, the system waited three minutes before collecting data in an attempt to allow the voltage readings from the rods to stabilize. After three minutes, the Arduino script instructs the ADC to perform ten differential measurements--one every second--between ports A0 (positive, Titanium) and A1 (negative, Zinc), perform a conversion of these digital values to millivolts, and send the values through the serial port. By taking differential measurements between two rods rather than single-ended measurements on both rods, the Zinc rod acts as an effective “ground,” and the value output by the ADC is the potential difference between rods, rather than two separate floating potentials.

Data analysis was performed in a Matlab script--a prompt asks the user whether a new sample is ready to be tested, the user enters the "true" pH of the sample, obtained by the electronic pH meter. During preliminary testing and up until the full final product and all of its subsystems were completely integrated, the Feather Huzzah was connected directly to a computer via USB, allowing MATLAB to rapidly read large numbers of data points via the serial port. After the final product was complete and assembled, data was sent wirelessly to ThingSpeak, and so rather than reading data from the serial port, the MATLAB script used the ThingSpeak API to retrieve the data that had been sent. After testing as many samples as desired, the script manipulates the voltage difference data to calculate averages for each sample and graph these averages against the samples' "true" pH levels.

Results

The results from preliminary testing were used to create a linear equation linking measured voltage difference between rods to a specific pH. However, during testing that took place after the full system had been assembled, we found that the voltage difference values being obtained

by the rods occurred across a much narrower range than they had been during preliminary tests (roughly 350 mV - 600 mV vs the previous 200 mV - 750 mV). There are several possible causes of this phenomenon. The first is that the solutions used for preliminary testing were created using pH Up and pH Down, which was later discovered to be very unstable. Another possible cause is that as testing progressed, the rods began to show signs of wear as they were sanded and subject to stressful conditions (one rod became slightly bent during installation of a separate subsystem). Any change in rod geometry could have a significant effect on the voltages being read. A final hypothesis is that waiting for three minutes before beginning data collection was too long. During proof-of-concept tests earlier in the semester, it was observed that voltage readings in lower-pH solutions stabilized more quickly, while readings in more neutral-pH solutions climbed steadily over time (Figure 9). This could cause the range of voltage difference values to

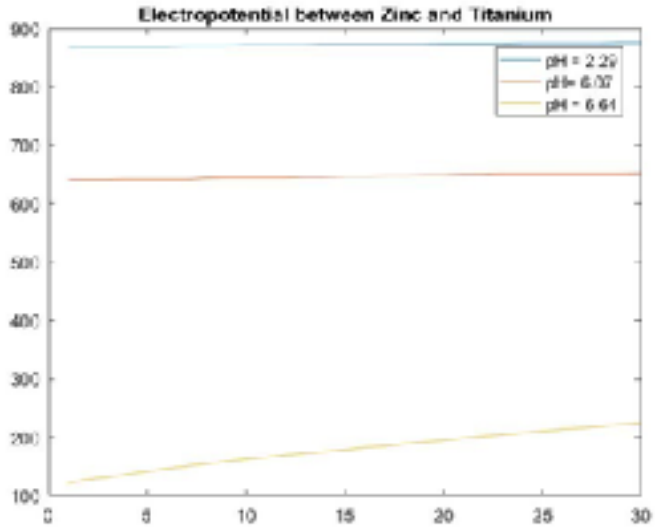


Figure 9. Proof-of-concept testing results for three different pH levels

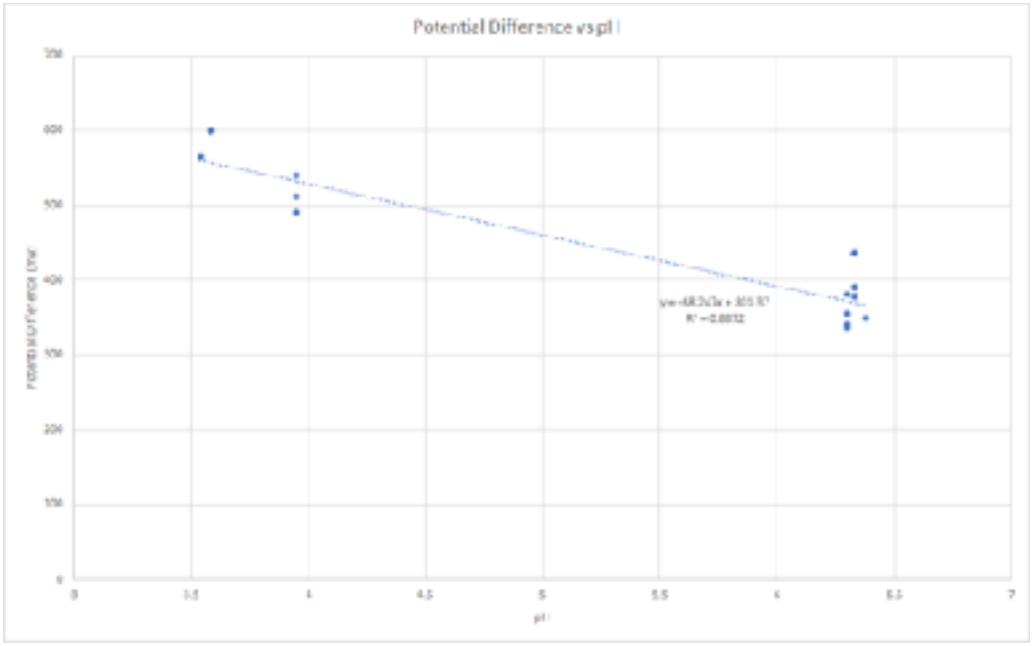


Figure 10. Voltage Difference vs pH, created using system fully assembled.

decrease as the readings climbed closer to the higher voltage differences characteristic of low-pH solutions.

Upon noticing this decrease in value range, a new linear equation was created (Figure 10). This new set of data points contained much higher standard deviations than previous sets, but due to time constraints we moved forward with the new linear equation. Unfortunately, due to these large standard deviations, we did not feel overly confident in our system's pH predictions when looking at isolated testing instances. However, the linear equation could still function correctly if the final demo permitted the system to be dunked and new measurements to be taken about five times and averaged together in an attempt to compensate for the high standard deviations. We believe these standard deviations are caused by wear and imperfections on the rods, as well as the fact that only a small amount of each rod was exposed to solutions, possibly limiting the levels of interaction the rods could have with ions in the solutions. Given more time, we would be interested to explore adding more of the rods and creating a more sophisticated prediction algorithm--we feel that the extra rods could provide levels of redundancy could help compensate for rod imperfections. Also, testing with a wider variety of samples also would allow us to create a more accurate linear equation.

V. Wireless Communication

Background & Design Goals

The overall task for wireless communication was to transfer data collected by the sensors connected to the Feather Huzzah to an external computer, which could then perform analysis. To complete this task, we kept in mind a few critical constraints: (1) our system would need to be disconnected from the Wi-Fi when placed underwater, (2) the device could not be touched after leaving the water, and (3) data should be easily accessible and analyzed on an external computer. To meet these constraints, we decided to use various technologies and applications in our wireless design plan, always prioritizing ease of use and reliability.

Overall Data Flow

Figure 11. Wireless communication system overview, split into two main parts: before data collection and after data collection.

We divided our wireless communication plan into two distinct parts to simplify the process (Figure 11). The first part, everything that happens before data collection or the device being submerged, involved mainly finding a way to send a signal to the Feather Huzzah before starting a test. The second part included everything after the system was submerged in water: data collection, transmission and analytics.

Before Data Collection

To send a signal to our Feather Huzzah, we used TalkBack, an application built into ThingSpeak that facilitates sending simple HTTP requests, which is discussed later. Once an HTTP string is created in the TalkBack application, the Feather Huzzah, while connected to WiFi, could check every 5 seconds for the command, at which point it would begin the test.

Data Collection, Transmission & Analytics

The second part of our wireless plan contains everything during and after data collection, summarized in Figure 12. Since we found our readings from the sensors and ADC to be consistent with low standard deviations we decided to limit the number of points used per test and no longer include the SD card and SD card module we had planned on using initially for local storage. After the initial TalkBalk signal is set to the Feather Huzzah, the system waits 3 minutes before taking any readings to allow the electrochemical rods and water to settle. After 3

minutes, the system takes one data point for one of the color sensors and voltage every 5 seconds. Because our system used two of the same Adafruit TCS34725 color sensors, we were unable to read from the sensors simultaneously because the SCL and SDA port IDs were the same. To resolve this we switched off reading from one sensor to the other halfway through the 30 second data collection period. All data collected was merged into a single JSON string buffer in the format specified by ThingSpeak's bulk update API discussed later.

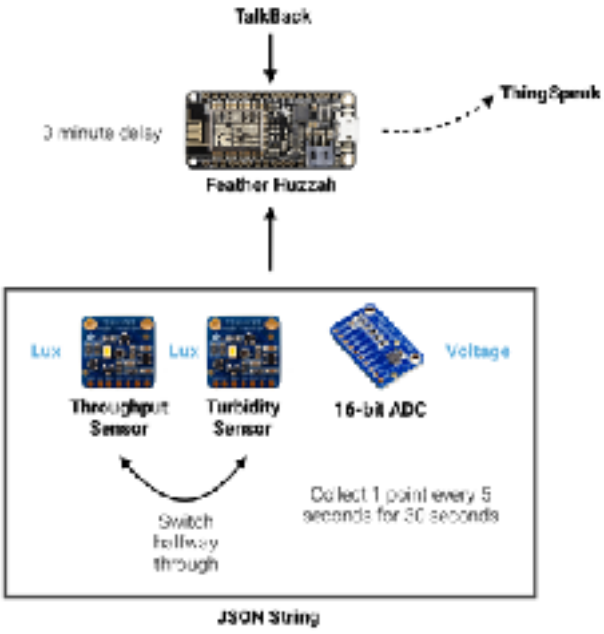


Figure 12. Data collection and transmission overview

Once the total 3 minutes and 30 seconds of testing is complete and the Feather Huzzah is un-submerged from the water, the system reconnects to WiFi and transmits the JSON string buffer to ThingSpeak and clears the buffer for reuse. Once the data is in ThingSpeak, we run our MATLAB code to analyze our data as will be discussed later.

TP-Link Nano

A TP-Link Wireless N Nano Router was given to us to serve as a wireless access point between the Feather Huzzah and computer [11]. The setup for this device was very straightforward and provided a strong and reliable wireless signal. Reconnecting to the router after disconnecting was also very quick, which is great for a design that requires constant reconnecting like ours.

ESP8266

The ESP8266 wireless chip built into the Feather Huzzah was also very straightforward to set up and had easy-to-use libraries built into Arduino [12]. In our brief tests the chip was able to disconnect and reconnect to the wireless access point quickly and reliably. As predicted the chip also loses Wi-Fi connection when dunked under water, but with our SD card solution for local storage this should not be an issue.

ThingSpeak

To act as our online server to push data to and query data from we decided to use ThingSpeak, a website built by Mathworks to allow easy communication between sensors and online servers. Additionally, ThingSpeak provides easy-to-use APIs to analyze collected data directly in MATLAB on a local computer.

```
"{\"write_api_key\":\"YOUR-CHANNEL-
WRITEAPIKEY\",\"updates\":
[{\"delta_t\":0,\"field1\":-60},
{\"delta_t\":15,\"field1\":200},
{\"delta_t\":15,\"field1\":-66}]''

delta_t = automatically updated clock
variable
```

Figure 13. JSON Buffer format for ThingSpeak bulk update API

ThingSpeak only allows data to be sent to its servers every 15 seconds, which we deemed as a major problem in our final design [13]. To fix this problem we switched our code to use ThingSpeak's bulk update APIs that allowed a device to collect unlimited amounts of data and upload it all at once. To use the APIs we needed to append all the data values the sensors were reading to a JSON string buffer with the format shown in Figure 13. Once data was ready to be sent, a HTTP request needed to be sent to the ThingSpeak server with our specific channel ID and API keys in order for the data to transmit successfully.

A channel has been set up on ThingSpeak for all of our group's data. Each channel in ThingSpeak is split into eight fields that store data separately. Fields 1-4 correspond to red, green, blue, and lux readings from the throughput sensor. Fields 5-7 correspond to red, green, blue readings from the turbidity sensor (lux had to be calculated separately because of the 8 field limit per channel). Field 8 held all voltage readings from our ADC. Note that data can be transmitted to whichever fields are needed at any given time. For example, for our final in-class demonstration, since turbidity was not a factor, we stopped collecting data for fields 5-7 to speed up testing, and ThingSpeak was able to correctly handle missing entries.

TalkBack

To send a start signal easily to our Feather Huzzah before testing, we used ThingSpeak's built-in application: TalkBack [13]. TalkBack allowed us to send it an HTTP string, which it would store as a command on its servers until a device sent an HTTP get request to retrieve it. Our system

only used one “start” signal to begin testing and continued to use further “start” signals to begin another test after the first had concluded. Similar to ThingSpeak, TalkBack had easy to use examples and API which made integration into our system very simple.

Figure 14. Example output from MATLAB script showing automatically ordered samples, pH estimates and generated plots

MATLAB Analytics

To make data analytics easy, we transfer all data from ThingSpeak to MATLAB after all samples are run. Using the API ThingSpeak provides, MATLAB can collect as many of the latest data points from any field in any channel as specified. Since the Feather Huzzah is only uploading to ThingSpeak after every test with a specific number of samples and data points, we were able to automate the plotting and analytics. A field at the top of the MATLAB script holding the number of samples tested can be changed to specify the number of data points to be read from ThingSpeak. The script automatically plots mean data for both color sensors and ADC output. It also orders the samples in terms of relative Rhodamine WT dye concentration based on mean lux values, and estimates pH for each sample using a manually inputted best fit equation created after extensive testing. An example output from our MATLAB script from our in-class demonstration is shown in Figure 14.

VI. System Integration

Background

System integration of the different sensors into an automatic system was a complex aspect of this project. Due to the sensitivity of electronics to water, the electronics were not fully integrated into the system until the housing was proven to be watertight. However, creating a watertight housing that also allowed access for development proved to be a challenge - and very quickly the

extra credit challenge associated with the high pressure rating was abandoned. In the end, a simple 4" PVC pipe was used to create a pill-like shape with a union joint in the middle for easy access [15]. In future development, the design could have a similar form factor but be miniaturized.

Design Goals

Three main goals governed the system integration design:

Easy Access to Electronics

Due to the prototypical nature of the project, it was necessary to have easy access to the electronics. This meant that the electronics could not be epoxied or potted in any way (which would solve the waterproofing issue). It also meant that the design had to be large enough for manipulation of the system once electronics were installed.

Watertight enclosure

The most important part of the housing was that it protected the electronics that were internally held. This was complicated by two further factors: firstly, the colorimetry sensor required that the water flow into the system so that throughput and turbidity readings could take place. Additionally the requirement to have access to the electronics meant that a removable seal was needed.

Subsystem Integration

Each of the various subsystems need access to the Huzzah Breakout board and the same battery power. If the system is to come separate for access, then the wire management is an important design consideration.

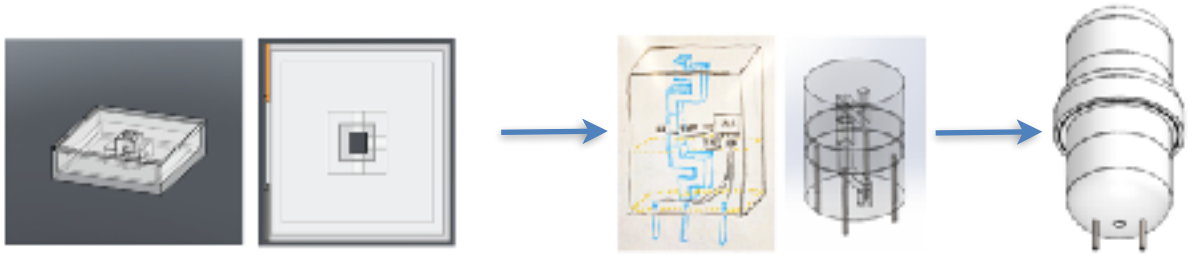


Figure 15. Evolution of the product packaging, from 3D-printed box up to the final PVC design.

Testing & Revisions

The electronics within this device essentially did not change once the sensors were created. The system design changed considerably during the semester as we tried to fulfill the main design considerations and completed several weeks of waterproof testing. The overall housing went through a series of major design changes and eventually settled on the final, PVC-based design due to ease of use and manufacture.

Within the final external housing the colorimetry sensor housing went through number of changes as a turbidity sensor was incorporated, and to allow reliable and consistent geometry within the sensor set up (Figures 1 and 2, above). All iterations were 3D printed.

Additionally, the system which enabled the water to flow through the colorimetry sensor changed - initially a cuvette-to-hose system with 3D printed connectors (Figure 16a). However, as the initial system was extremely unreliable during waterproof testing - a standard pipe was used instead with an interlocking piece of smaller pipe, which proved sufficiently accurate for the task (Figure 16b).

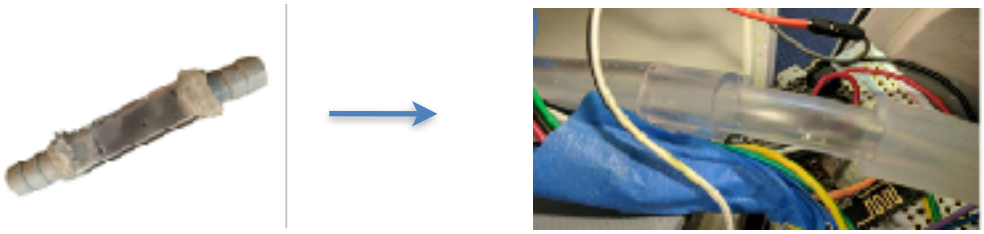


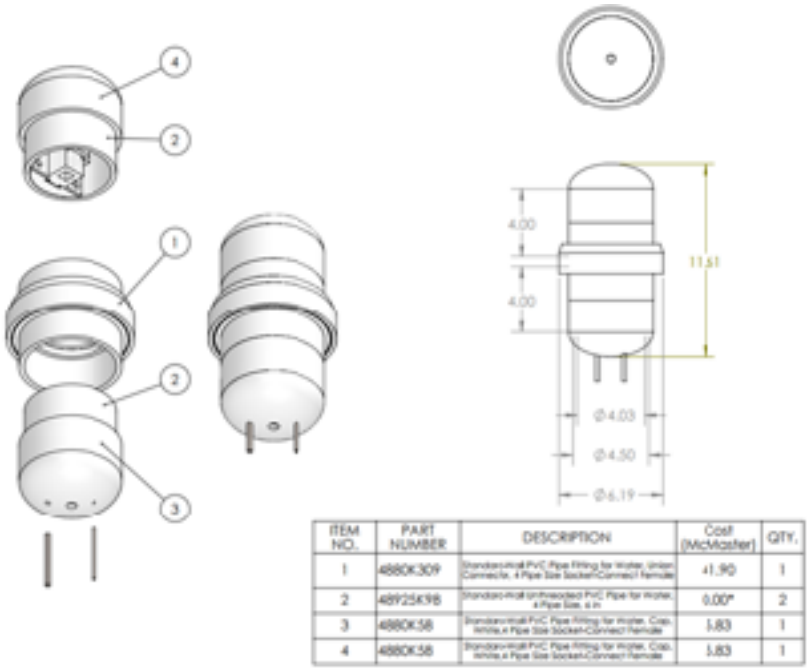
Figure 16. (a) 3D-printed tube-to-cuvette interface. (b) Final, tubing-only connection.

Final Design

The final design of our system is a modular pill-shape PVC pipe design with major dimensions of 11.61" x 6.19". The parts were chosen to ensure reliability and ease-of-access. The full final design was able to be modeled in Solidworks, which allowed for visualization and quick iteration of internal 3D printed parts (Figure 17).

The pipe was made up of two half sections, joined at the center with a union joint (a screw down part with an incorporated O-ring). All other potential water entry points were sealed with Loctite Marine Fast Cure Sealant.

The physical pH system was made up of two protruding rods which were sealed in place with epoxy. The colorimetry system was a clear pipe that runs the entire length of the system. It



ITEM NO.	PART NUMBER	DESCRIPTION	Cost (McMaster)	QTY.
1	4880K309	Standard-Wall PVC Pipe Fitting for Water, Union Connector, 4 Pipe Size Socket/Connect Female	41.90	1
2	48925K98	Standard-Wall Unthreaded PVC Pipe for Water, 4 Pipe Size, 4 in	0.00*	2
3	4880K58	Standard-Wall PVC Pipe Fitting for Water, Cap, White, 4 Pipe Size Socket-Connect Female	1.83	1
4	4880K58	Standard-Wall PVC Pipe Fitting for Water, Cap, White, 4 Pipe Size Socket-Connect Female	1.83	1

Figure 17. Final Solidworks design schematics (with BOM)

Figure 18. Final physical design

is possible to split the pipe into two sections when taking system apart due to a smaller pipe acting as a male-male connection.

Retrospectively, the design was probably a little large for the sensor system we were trying to create - future designs should keep same form factor but reduce the pipe diameter. However, the system's size enabled simple access to the insides of the system, which is convenient for making repairs or alterations.

VII. Power Consumption & Battery Life

The power consumption estimates for all the electronics in our system are listed in Table 1. The 3.7V battery we use for our design runs on 6600 mAh, or 6.6Ah. The battery's current hours divided by our system's current draw, 6.6Ah / 0.50148A, gives us an absolute minimum battery life of approximately 13.16 hours. Since our system takes approximately 3.5 minutes per test, it can run at least 225 tests on one single battery charge. Note that the estimates provided in Table 1 are peak currents, so we expect that the true battery life of our system be slightly greater than 13.16 hours.

Table 1: Power Consumption Estimate

Part	Peak Current (A)	Voltage Drawn (V)	Parts Used	Power (W)
Adafruit Feather Huzzah w/ ESP8266	0.5	3.7	1	1.85
Adafruit 16-bit ADC	0.00015	3.7	1	0.000555
Adafruit TCS34725	0.00033	3.3	2	0.002178
Adafruit NeoPixel RGBW LEDs	0.001	3.7	1	0.0037
Total Current (A)	**0.50148**		**Total Power (W)**	**1.856433**

VIII. Environmental Impact Statement

The environmental impact of this physical design is minimal - electronics, and PVC/ABS plastics were used, which, if lost during field testing, are non-toxic pollutants. Do to the procedures the system performs being relatively unobtrusive, the system will not physically damage the environment if used correctly.

The system will be tracing Rhodamine WT which (in high enough concentrations) can be deadly to fish and molluscs. Therefore the concentrations at which we were testing during this semester should not be replicated within actual environments. However, the precision of our colorimeter system should allow the system to function well at lower, safer concentrations of Rhodamine WT.

The pH sensor component of this device has very little environmental impact, and the data from the pH sensor can be used to attempt to answer important question pertaining to different areas of study such as ocean acidification and climate change.

IX. Cost Analysis

A breakdown of all of our costs in prototyping our device are listed in Tables 2 and 3. A breakdown of costs for manufacturing 10000 of our devices is presented in Table 4. Note that some of the parts costs such as for the zinc and titanium rods needed to be estimated using prices online. For most other parts however, costs for purchasing in bulk were found directly on manufacturer or reseller websites. If no bulk prices were found online for manufacturers such as McMaster-Carr, the original single unit price was used, which means the true manufacturing cost per device may be slightly lower than reported in Table 4.

Table 2: Prototyping Parts Costs

Part	**Manufacturer**	**Used in final product?**	**Price**
LED Breakout Board	SparkFun	No	$2.95
TCS34725 RGB Color Sensor x3	Adafruit Industries	Yes (2)	$23.85
NeoPixel Mini RGB LED x10	Adafruit Industries	No	$4.95
NeoPixel RGBW LEDs x10	Adafruit Industries	Yes (1)	$5.95
TSL257 Photodetector	TSL	No	$3.27
32-Ounce Mixing Cups x12	Auto Body Now	No	$10.95
4” PVC Plug	Charlotte Pipe	No	$2.28
4” PVC Union Connector	McMaster-Carr	Yes	$41.90
4” PVC Cap x2	McMaster-Carr	Yes	$11.66
Vinyl Tube	Home Depot	Yes	$8.87
Saddle Connector Clamps x10	Flex	Yes	$4.25
SD Card Module	Adafruit Industries	No	$8.95
		Total Parts Cost	**$129.83**

Table 3: Prototyping Shipping/Tax Costs

Vendor	**Shipping/Tax Costs**
Adafruit Batch 1	$9.09
Adafruit Batch 2	$9.11
Sparkfun	$5.71
Digikey	$2.85

Vendor	Shipping/Tax Costs
Home Depot	$1.26
McMaster-Carr	$7.39
Adafruit	$9.11
Total Shipping/Tax Cost	**$44.52**

Total Prototyping Cost: $174.35

Table 4: Estimated Costs of Manufacturing 10000 Devices

Part	Manufacturer	Number Required	Unit Price	Total Price
TCS34725 RGB Color Sensor	Adafruit Industries	20000	$6.36	$127,200.00
NeoPixel RGBW LEDs x10	Adafruit Industries	1000	$4.76	$4,760.00
4" PVC Union Connector	McMaster-Carr	10000	$41.90	$419,000.00
4" PVC Cap x2	McMaster-Carr	5000	$11.66	$58,300.00
Vinyl Tube (good for 10 devices)	Home Depot	1000	$8.87	$8,870.00
Saddle Connector Clamps x10	Flex	1000	$4.25	$4,250.00
Feather Huzzah w/ ESP8266	Adafruit Industries	10000	$13.56	$135,600.00
Lithium Ion Battery Pack - 3.7V, 6600 mAh	Adafruit Industries	10000	$23.60	$236,000.00
Zinc Rod 2/8" x 36" (2 inches used per device)	Boat Zincs	556	$14.82	$8,239.92
Titanium Rod 2/8" x 34.8" (2 inches used per device)	TMS Titanium	575	$57.00	$32,775.00
ADS1115 16-bit ADC	Adafruit Industries	10000	$11.96	$119,600.00
Perma-Proto Quarter-sized Breadboard x3	Adafruit Industries	3334	$6.80	$22,671.20
3D Printed Material (estimated)	3D Printed	10000	$5.00	$50,000.00
4" x 120" PVC Pipe (7 inches used per device)	Charlotte Pipe	584	$16.04	$9,367.36

Breadboard Jumper Wires x 75 (5 used per device)	Dragonpad	667	$5.90	$3,935.30
			Total Price	**$1,240,568.78**
			Price Per Device	**$124.06**

X. Conclusion

At the end of the semester, we have created a watertight pH- and Rhodamine WT relative concentration-sensing suite. We are very pleased with the robustness and accuracy of the colorimeter subsystem, which performed consistently in the tested samples and is able to perform accurately without alterations even in low-turbidity solutions. It was also powerful enough to be just as effective when taking readings through a clean and clear cuvette as when taking readings through a clear but thick cylindrical tube.

The code written for this project, including wireless communication, Arduino, and MATLAB, is efficient, user-friendly, well-commented, and very easy to understand and alter. The wireless communication system sends data over Wi-Fi without any loss, and is well-equipped to wait to send data in the case of signal loss until connection has been reestablished.

The inside of the system is very easy to access while still remaining watertight. Such an open system is extremely valuable because of the inevitable possibility of individual component failure, as well as the possibility of other repairs or modifications that could extend the project further.

While the subsystem's design is simple and intuitive, we would have like to spend more time modifying the pH-sensing unit. Using additional rods, more sophisticated curve fitting or weighting of data points, and simply testing the system on a wider range of solutions would likely allow us to improve this subsystem. However, we are pleased that we identified trends in the data and were able to generally match later tests to those trend-lines.

Despite the underwhelming pH results, we feel that we have assembled a solid, reliable, and accessible sensing suite with a powerful colorimeter for Rhodamine WT relative dye concentration and a reliable wireless communication system.

XI. References

[1] “Shell Ocean Discovery XPRIZE", Shell Ocean Discovery XPRIZE, 2017. [Online]. Available: http://oceandiscovery.xprize.org. [Accessed: 26- Apr- 2017].

[2] A. Industries, "Adafruit Feather HUZZAH with ESP8266 WiFi ID: 2821 - $16.95 : Adafruit Industries, Unique & fun DIY electronics and kits", Adafruit.com, 2017. [Online]. Available: https://www.adafruit.com/product/2821. [Accessed: 26- Apr- 2017].

[3] J. Wilson, E. Cobb and F. Kilpatric, Fluorometric Procedures for Dye Tracing, 1st ed. Washington, D.C.: United States Government Printing Office, 1986, pp. 8-10.

[4] Thermallaminatingfilms.com, 2017. [Online]. Available: http://www.thermallaminatingfilms.com/haze.php. [Accessed: 26- Apr- 2017].

[5] Adafruit. “NeoPixel RGBW LEDs w/ Integrated Driver Chip.” [Online]. Available: https://www.adafruit.com/product/2759, [Accessed: Mar. 24, 2017].

[6] Adafruit. “RGB Color Sensor with IR Filter and White LED.” [Online]. Available: https://www.adafruit.com/product/1334, [Accessed: Mar. 24, 2017].

[7] “Corrosion of Metals.” [Online]. Available: http://xapps.xyleminc.com/Crest.Grindex/help/grindex/contents/Metals.htm, [Accessed: Mar. 23, 2017].

[8] M. Finšgar, "Galvanic series of different stainless steels and copper- and aluminium-based materials in acid solutions", Corrosion Science, vol. 68, pp. 51-56, 2013.

[9] M.A. Brooke et al. “An Ocean Sensor for Measuring the Seawater Electrochemical Response of 8 Metals Referenced to Zinc, for Determining Ocean pH.” in ICST 2015, Dec. 2015.

[10] Adafruit. “ADS1115 16-Bit ADC.” [Online]. Available: https://www.adafruit.com/product/1085, [Accessed: Mar. 23, 2017].

[11] "150Mbps Wireless N Nano Router - TP-Link", Tp-link.com.au, 2017. [Online]. Available: http://www.tp-link.com.au/products/details/TL-WR702N.html. [Accessed: 24- Mar- 2017].

[12] "WiFi Module - ESP8266 - WRL-13678 - SparkFun Electronics", Sparkfun.com, 2017. [Online]. Available: https://www.sparkfun.com/products/13678. [Accessed: 24- Mar- 2017].

[13] "How to Buy - ThingSpeak", Thingspeak.com, 2017. [Online]. Available: https://thingspeak.com/prices. [Accessed: 24- Mar- 2017].

[14] D. Kim and H. Kim, "Waterproof characteristics of nanoclay/epoxy nanocomposite in adhesively bonded joints", Composites Part B: Engineering, vol. 55, pp. 86-95, 2013.

[15] C. Staff, "Everything You Need To Know About PVC Plastic", Creativemechanisms.com, 2017. [Online]. Available: https://www.creativemechanisms.com/blog/everything-you-need-to-know-about-pvc-plastic. [Accessed: 24- Mar- 2017].

XII. Appendix

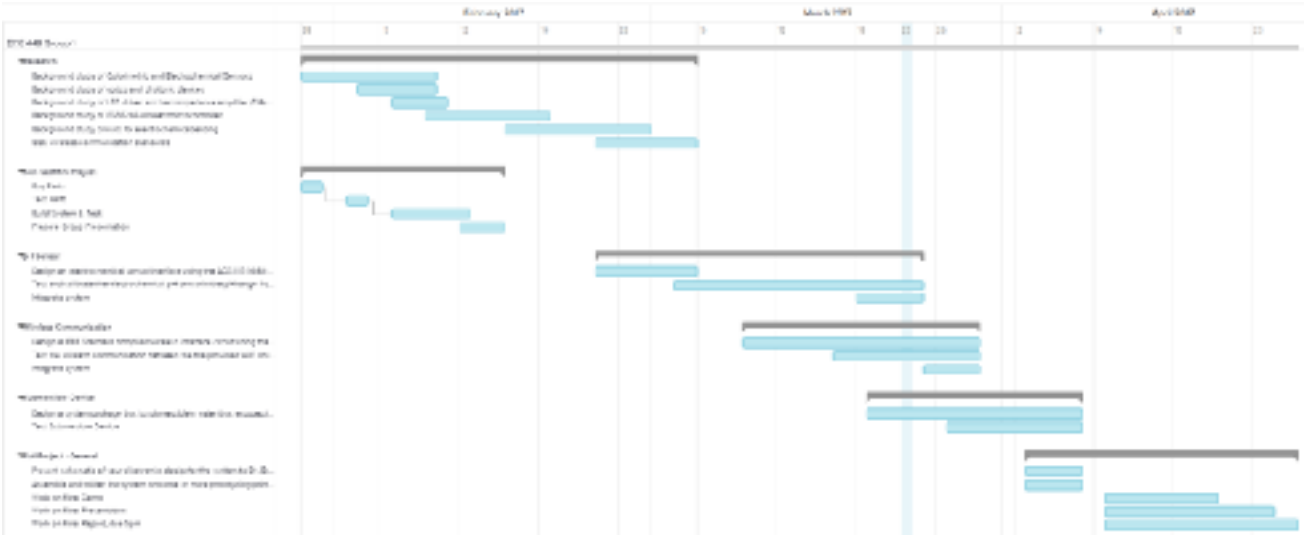


Figure A1: Gantt Chart

Arduino Code:

```
#include <Wire.h>
#include <Adafruit_ADS1015.h>
#include <Adafruit_TCS34725.h>
#include <Adafruit_NeoPixel.h>
#include <ThingSpeak.h>
#include <EthernetClient.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// Tunable global variables
char ssid[] = "449Group1"; //  Your network SSID (name)
char pass[] = "wedemboyz"; // Your network password
const unsigned long postingInterval = 30L * 1000L; // Post data every 100 seconds (don't go over
100)
const unsigned long updateInterval = 3L * 1000L; // Update once every 15 seconds (don't go below
10)
const int checkTalkBackInterval = 10 * 1000;    // Time interval in milliseconds to check TalkBack
(number of seconds * 1000 = interval)
const int submergeTime = 180 * 1000; // Time for device to be submerged in water before starting
readings
int sensorA = 13;  // digital pin 13 powers sensor A
```

```
int sensorB = 15;  // digital pin 15 powers sensor B
int LEDpin = 12;   // digital pin 12 powers the LED

// Initialize 16-bit ADC
Adafruit_ADS1115 ads;

// Initialize the jsonBuffer to hold data
char jsonBuffer[500] = "[";

// Initialize clock variables
unsigned long lastConnectionTime = 0; // Track the last connection time to ThingSpeak channel
unsigned long lastUpdateTime = 0; // Track the last update time

// ThingSpeak Server
char server[] = "api.thingspeak.com";
String thingSpeakAPI = "api.thingspeak.com";

// Initialize sensors & variables
int SensorActivePin = sensorA; // Activate sensor A to begin with
boolean switchSensor = true; // True when first 2 data points need to be skipped
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);
uint16_t r, g, b, c, lux = 0;
int16_t voltageReading;
float multiplier = 0.1875F; // ADS1115  @ +/- 6.144V gain (16-bit results) //

// Initialize LED
Adafruit_NeoPixel strip = Adafruit_NeoPixel(1,LEDpin);

void setup() {
  Serial.begin(115200);

  // Start LED
  strip.begin();
  strip.setPixelColor(0,255,255,255,0);
  strip.show();

  // Start ADC
  ads.begin();

  // Start RGB sensors
  pinMode(sensorA, OUTPUT);  // set pin to power sensorA
  pinMode(sensorB, OUTPUT);  // set pin to power sensorB
  digitalWrite(sensorB, LOW);
  digitalWrite(sensorA, HIGH);

  // Attempt to connect to wifi
  attemptWifi();

  // Check TalkBack for start signal
  checkTalkBack();
}

void loop() {
  // If halfway through updates, switch sensors
```

```
  if (((millis()- lastConnectionTime) >=  postingInterval/2) && (SensorActivePin == sensorA)) {
    sensorSwitch();
  }

  // If update time has reached 15 seconds, then update the jsonBuffer
  if (millis() - lastUpdateTime >=  updateInterval) {
    updatesJson(jsonBuffer);
  }
}

// Updates the jsonBuffer with data
void updatesJson(char* jsonBuffer) {
  /* JSON format for updates paramter in the API
   *  This example uses the relative timestamp as it uses the "delta_t". If your device has a real-
time clock, you can provide the absolute timestamp using the "created_at" parameter
   *  instead of "delta_t".
   *  "[{\"delta_t\":0,\"field1\":-70},{\"delta_t\":15,\"field1\":-66}]"
   */

  // Reinitialize sensor every time function is run
  Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);

  // If reading from the sensor for the first time since a switch, skip the first two readings
  if (switchSensor == true) {
    for(int i = 0; i < 2; i++){
      tcs.getRawData(&r, &g, &b, &c);
      delay(200);
    }
    switchSensor == false;
  }

  // Get sensor readings
  tcs.getRawData(&r, &g, &b, &c);
  lux = tcs.calculateLux(r, g, b);

  Serial.print(lux, DEC); Serial.print(" ");
  Serial.print(r, DEC); Serial.print(" ");
  Serial.print(g, DEC); Serial.print(" ");
  Serial.print(b, DEC); Serial.println(" ");

  // Get voltage reading
  voltageReading = ads.readADC_Differential_0_1();
  int voltage = voltageReading * multiplier;
  Serial.println(voltage);

  // Initialize variables for jsonBuffer
  char temp[4];
  unsigned long deltaT = (millis() - lastUpdateTime)/1000;
  size_t lengthT = String(deltaT).length();

  // If sensorA is active, add to fields 1-4,8
  if (SensorActivePin == sensorA) {
    strcat(jsonBuffer,"{\"delta_t\":");
    String(deltaT).toCharArray(temp,lengthT+1);
```

```
    strcat(jsonBuffer,temp);

//    //Add R to JsonBuffer
//    strcat(jsonBuffer,",");
//    strcat(jsonBuffer, "\"field1\":");
//    lengthT = String(r).length();
//    String(r).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);
//
//    //Add G to JsonBuffer
//    strcat(jsonBuffer,",");
//    strcat(jsonBuffer, "\"field2\":");
//    lengthT = String(g).length();
//    String(g).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);
//
//    //Add B to JsonBuffer
//    strcat(jsonBuffer,",");
//    strcat(jsonBuffer, "\"field3\":");
//    lengthT = String(b).length();
//    String(b).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);

    //Add Lux to JsonBuffer
    strcat(jsonBuffer,",");
    strcat(jsonBuffer, "\"field4\":");
    lengthT = String(lux).length();
    String(lux).toCharArray(temp,lengthT+1);
    strcat(jsonBuffer,temp);

    //Add Voltage to JsonBuffer
    strcat(jsonBuffer,",");
    strcat(jsonBuffer, "\"field8\":");
    lengthT = String(voltage).length();
    String(voltage).toCharArray(temp,lengthT+1);
    strcat(jsonBuffer,temp);

    strcat(jsonBuffer,"},");
  }
  // If sensorB is active, add to fields 5-8
  else {
    strcat(jsonBuffer,"{\"delta_t\":");
    String(deltaT).toCharArray(temp,lengthT+1);
    strcat(jsonBuffer,temp);

//    //Add R2 to JsonBuffer
//    strcat(jsonBuffer,",");
//    strcat(jsonBuffer, "\"field5\":");
//    lengthT = String(r).length();
//    String(r).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);
//
//    //Add G2 to JsonBuffer
//    strcat(jsonBuffer,",");
```

```
//    strcat(jsonBuffer, "\"field6\":");
//    lengthT = String(g).length();
//    String(g).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);
//
//    //Add B2 to JsonBuffer
//    strcat(jsonBuffer,",");
//    strcat(jsonBuffer, "\"field7\":");
//    lengthT = String(b).length();
//    String(b).toCharArray(temp,lengthT+1);
//    strcat(jsonBuffer,temp);

    //Add Voltage to JsonBuffer
    strcat(jsonBuffer,",");
    strcat(jsonBuffer, "\"field8\":");
    lengthT = String(voltage).length();
    String(voltage).toCharArray(temp,lengthT+1);
    strcat(jsonBuffer,temp);

    strcat(jsonBuffer,"},");
  }

  // Update the last update time
  lastUpdateTime = millis();

  // If 100 seconds has passed, get ready to send data and reset
  if ((millis() - lastConnectionTime) >=  postingInterval) {
    attemptWifi(); // Connect to WiFi
    size_t len = strlen(jsonBuffer);
    jsonBuffer[len-1] = ']';
    httpRequest(jsonBuffer); // Send data to ThingSpeak
    sensorSwitch(); // Switch active sensor back to sensorA
    checkTalkBack(); // Wait for next start signal
  }
}

// Switch active sensor
void sensorSwitch(){
  Serial.println("Sensor switch");
  switchSensor = true;
  digitalWrite(SensorActivePin, LOW);
  if (SensorActivePin == sensorA) {
    SensorActivePin = sensorB;
  }
  else {
    SensorActivePin = sensorA;
  }
  digitalWrite(SensorActivePin, HIGH);
  delay(5000);
}

// Check TalkBack for start signal
void checkTalkBack() {
  HTTPClient http;
```

```
  String talkBackCommand;

  // Continue running infinitely until signal is received
  while (1) {
    char charIn;
    String talkBackURL =  "http://" + thingSpeakAPI + "/talkbacks/" + 14958 + "/commands/execute?
api_key=" + "61QGI8MFYN17D3VY";

    // Make a HTTP GET request to the TalkBack API:
    http.begin(talkBackURL);
    int httpCode = http.GET();
    Serial.print("Waiting for user input, ");
    Serial.print("httpCode: ");
    Serial.println(httpCode>0);

    talkBackCommand = http.getString();
    http.end();

    // Start script if triggered
    if (talkBackCommand.equals("Start")) {
      Serial.println(talkBackCommand);
      delay(submergeTime);

      // Restart clock variables
      lastUpdateTime = millis();
      lastConnectionTime = millis();
      return;
    }
    delay(checkTalkBackInterval);
  }
}

// Attempt to connect to WiFi network
void attemptWifi(){
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, pass);  // Connect to WPA/WPA2 network. Change this line if using open or WEP
network
    delay(5000);  // Wait 5 seconds to connect
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("Failed connecting to wifi");
    }
    else {
      Serial.println("Connected to wifi");
    }
  }
  printWiFiStatus(); // Print WiFi connection information
}

// Updates the ThingSpeakchannel with data
void httpRequest(char* jsonBuffer) {
  /* JSON format for data buffer in the API
```

```
   *  This example uses the relative timestamp as it uses the "delta_t". If your device has a real-
time clock, you can also provide the absolute timestamp using the "created_at" parameter
   *  instead of "delta_t".
   *   "{\"write_api_key\":\"YOUR-CHANNEL-WRITEAPIKEY\",\"updates\":[{\"delta_t\":
0,\"field1\":-60},{\"delta_t\":15,\"field1\":200},{\"delta_t\":15,\"field1\":-66}]
   */
  // Format the data buffer as noted above
  char data[500] = "{\"write_api_key\":\"M370S66LE099HJPD\",\"updates\":"; //Replace YOUR-CHANNEL-
WRITEAPIKEY with your ThingSpeak channel write API key
  strcat(data,jsonBuffer);
  strcat(data,"}");

  // Initialize the WiFi client library
  WiFiClient client;

  String data_length = String(strlen(data)+1); //Compute the data buffer length
  // POST data to ThingSpeak
  if (client.connect(server, 80)) {
    client.println("POST /channels/244820/bulk_update.json HTTP/1.1"); //Replace YOUR-CHANNEL-ID
with your ThingSpeak channel ID
    client.println("Host: api.thingspeak.com");
    client.println("User-Agent: mw.doc.bulk-update (Arduino ESP8266)");
    client.println("Connection: close");
    client.println("Content-Type: application/json");
    client.println("Content-Length: "+data_length);
    client.println();
    client.println(data);
  }
  else {
    Serial.println("Failure: Failed to connect to ThingSpeak");
    delay(5000);
    httpRequest(jsonBuffer);
  }
  delay(250); //Wait to receive the response
  client.parseFloat();
  String resp = String(client.parseInt());
  Serial.println("Response code:"+resp); // Print the response code. 202 indicates that the server
has accepted the response
  if (!resp.equals("202")) {
    delay(10000);
    httpRequest(jsonBuffer);
  }
  jsonBuffer[0] = '['; // Reinitialize the jsonBuffer for next batch of data
  jsonBuffer[1] = '\0';
}

void printWiFiStatus() {
  // Print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // Print your device IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
```

```
  Serial.println(ip);

  // Print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

MATLAB Code:

```
%% Connection / Setup
clear all

readChannelID = 244820;
readAPIKey = '7PKO12RVT5C40PE8';
samplesTested = 3;
numPoints = (samplesTested*6)+samplesTested-1; % Number of data points to read from ThingSpeak

%% Data Collection
[data, time] = thingSpeakRead(readChannelID, 'NumPoints', numPoints, 'ReadKey', readAPIKey);

%% Parse data
dataEndA = 1;
dataEndB = 4;
dataEndVolt = 1;
for sample = 1:samplesTested
%     R(1:3,sample) = data(dataEndA:dataEndA+2, 1);
%     G(1:3,sample) = data(dataEndA:dataEndA+2, 2);
%     B(1:3,sample) = data(dataEndA:dataEndA+2, 3);
    Lux(1:3,sample) = data(dataEndA:dataEndA+2, 4);

%     R2(1:3,sample) = data(dataEndB:dataEndB+2, 5);
%     G2(1:3,sample) = data(dataEndB:dataEndB+2, 6);
%     B2(1:3,sample) = data(dataEndB:dataEndB+2, 7);

    Volt(1:6,sample) = data(dataEndVolt:dataEndVolt+5, 8);

    dataEndA = dataEndA+7;
    dataEndB = dataEndB+7;
    dataEndVolt = dataEndVolt+7;
end

%% Analyze
% meanR = mean(R);
% meanG = mean(G);
% meanB = mean(B);
meanLux = mean(Lux);
% meanR2 = mean(R2);
% meanG2 = mean(G2);
% meanB2 = mean(B2);
meanVolt = mean(Volt);
```

```
% stdR = std(R);
% stdG = std(G);
% stdB = std(B);
stdLux = std(Lux);
% stdR2 = std(R2);
% stdG2 = std(G2);
% stdB2 = std(B2);
stdVolt = std(Volt);

%% Plotting
x = 1:samplesTested;

figure(1)
clf
hold on
% p1 = plot(x,meanR,'r');
% errorbar(x,meanR,stdR,'vertical', 'r');
%
% p2 = plot(x,meanG,'g');
% errorbar(x,meanG,stdG,'vertical', 'g');
%
% p3 = plot(x,meanB,'b');
% errorbar(x,meanB,stdB,'vertical', 'b');

p4 = plot(x,meanLux,'k');
errorbar(x,meanLux,stdLux,'vertical', 'k');

ylabel 'Amplitudes'
xlabel 'Sample #'
title 'Color Intensity Means and Std. Deviations for 3 Points (Throughput Sensor)'

% legend([p1 p2 p3 p4],{'Red','Green','Blue','Lux'})
hold off

% figure(2)
% clf
% hold on
% p1 = plot(x,meanR2,'r');
% errorbar(x,meanR2,stdR2,'vertical', 'r');
%
% p2 = plot(x,meanG2,'g');
% errorbar(x,meanG2,stdG2,'vertical', 'g');
%
% p3 = plot(x,meanB2,'b');
% errorbar(x,meanB2,stdB2,'vertical', 'b');
%
% ylabel 'Amplitudes'
% xlabel 'Sample #'
% title 'Color Intensity Means and Std. Deviations for 3 Points (Turbidity Sensor)'

% legend([p1 p2 p3],{'Red','Green','Blue'})
% hold off

figure(3)
```

```
clf
hold on
p1 = plot(x,meanVolt,'k');
errorbar(x,meanVolt,stdVolt,'vertical', 'k');

ylabel 'Potential Difference (mV)'
xlabel 'Sample #'
title 'Potential Difference Means and Std. Deviations for 6 points'

hold off

%% Sample Sorting
for i = 1:length(meanLux)
    total(i) = -(meanLux(i));
end

[sorted, Index] = sort(total);

disp('Samples in order from lowest ppm to highest:')
for k = 1:length(meanLux)
    fprintf('Sample %d \n',Index(k))
end
fprintf('\n')

%% pH Estimation
pH_estimate1 = ((meanVolt(1)) - 801.37)./-68.247
pH_estimate2 = ((meanVolt(2)) - 801.37)./-68.247
pH_estimate3 = ((meanVolt(3)) - 801.37)./-68.247
```