Title

The Design and Testing of a Wireless Sensor Network for Real-Time Point of Use Water Quality Monitoring

Author

Fowler, McKenzie Lynn

Publication Date

2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

The Design and Testing of a Wireless Sensor Network for Real-Time Point of Use Water Quality Monitoring

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Science

in Civil Engineering

by

McKenzie Lynn Fowler

2021

ABSTRACT OF THE THESIS

The Design and Testing of a Wireless Sensor Network
for Real-Time Point of Use Water Quality Monitoring

by

McKenzie Lynn Fowler

Master of Science in Civil Engineering
University of California, Los Angeles, 2021
Professor Eric M.V. Hoek, Chair

Across the globe millions of people lack access to clean drinking water. This has led to widespread distrust of water resources, including tap water. One way to ensure the quality of drinking water and re-establish public trust is by implementing smart water monitoring technologies that can be utilized by operators and consumers. The work presented in this paper encompasses the design and testing of a wireless sensor network for point of use water quality monitoring. The final prototype allows for remote data monitoring and lays the foundations for further work to use the data acquired by the system for uses such as contamination detection and predictive modeling of water quality.

The thesis of McKenzie Lynn Fowler is approved

David Jassby

Jennifer Jay

Eric M.V. Hoek, Committee Chair

University of California, Los Angeles

2021

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I must thank my research advisor Dr. Eric M.V. Hoek. Without his valuable guidance, insight, and support I would not have been able to accomplish writing this thesis. I would also like to thank Dr. David Jassby and Dr. Jennifer Jay for serving on my thesis committee. All of whom have taught me so much and provided invaluable resources in my time at UCLA in their classes and throughout my research journey. In addition, I would like to thank all the members of my undergraduate research team who assisted throughout the project, your contributions are greatly appreciated. Deepest thanks must also go out to the sponsors of this research at 501CTHREE, who's vision and ongoing funding made this all possible. Lastly, I would like to acknowledge my family and friends. Words cannot express the gratitude I have for your constant support and encouragement in all my endeavors.

Introduction

As of 2010, safe and accessible drinking water has been explicitly recognized as a human right by the United Nations General Assembly and deemed essential for public health and economic prosperity. Despite this designation, 785 million people lack essential drinking water services globally, and at least two billion people use drinking water sources contaminated with feces. Diseases like cholera, diarrhea, dysentery, hepatitis, typhoid, and polio are linked to such sources of contamination (WHO, 2019). Despite the risks associated with a lack of essential water services, many drinking water utilities face challenges meeting regulations due to limited water supply, strict budgets, high demands due to population growth, aging infrastructure, and increasingly strict regulations for water quality. Analytical testing and water quality monitoring must be carried out regularly to ensure water is free from priority biological and chemical contamination. Unfortunately, these methods often take multiple days to conduct, in which case contaminated water may already have had negative impacts on public health.

These challenges can contribute to public water utilities violating regulations established by the EPA, increasing interest in solutions such as point-of-use (POU) filtration, and real-time water quality monitoring. Even if violations are not observed from water utilities there is still potential for contamination between treatment and the tap. This contamination can source back to community water system's distributional networks and property owner's premise plumbing (Pierce et al., 2019).

Before reaching taps, consumers can install POU drinking water treatment systems in their water supply lines to provide on-site treatment to the water they consume. These systems encompass many treatment technologies such as membranes, filtration, UV disinfection, activated carbon, etc. The utility of implementing POU filtration shows in its ability to reduce contamination's acute and chronic health effects. Still, there is little evidence of commercially

available technologies to improve the "smartness" of point-of-use water systems in terms of their ability to perform tasks such as monitoring and reporting on water quality (Wu et al., 2021.) Possible ways to enhance the smartness of point-of-use filters include integrating Internet of Things (IoT) enabled sensor technologies.

Commercially available sensors measure parameters including pH, oxidation-reduction potential (ORP), and electrical conductivity (EC), which can all be affected by chemical and biological contaminants—making these parameters effective indicators for the overall water quality as well as capable of detecting changes in water quality. When combined with the integrated circuit microcontrollers and micro processing technologies that have emerged in the last decade, there has become a platform for developing robust and customizable data loggers for relatively low costs. This fact combined with the application of data analysis techniques such as outlier detection and machine learning using back-end software allow for faster response times to possible contamination events, collection and analysis of large spatiotemporal data sets, and diagnostic capabilities for water treatment systems, as just a few of the potential benefits of real-time water quality monitoring.

The work presented in this manuscript describes the design, procurement, building, and testing of an IoT-enabled monitoring system prototype for POU applications with these goals in mind. The proposed system consists of a sensor package that includes basic water quality parameters of pH, EC, ORP, Dissolved Oxygen (DO), and temperature. The system uses a Raspberry Pi 4 Model B microprocessor. It includes a software program that enables data transfer from the sensors to an online repository where any collaborators who have access can view the data. Preliminary algorithms that utilize reference correlations, statistical analysis techniques, and physical and

chemical constants to calculate and predict other water quality parameters such as corrosivity risk have also been developed.

Related Work

Wireless Sensor Networks

Wireless sensor networks have been applied to an array of environmental monitoring problems. Those focused on water quality monitoring cover a range of applications for environmental waters, drinking water, and wastewater. Some primary questions addressed by previous work include lowering the costs of wireless sensor networks, optimizing power usage for proposed systems, developing software for contamination detection, and improving solutions for data transmission. The works presented utilize an array of programming languages, sensors, circuitry, and communication protocols that all serve to make real-time data monitoring accessible for operators and consumers.

Often environmental waters are prone to various perturbances, requiring the need for real-time monitoring. Rivers and marine coasts have been sites for the deployment of smart wireless sensor networks; Adu-Manu et al. (2020) employed wireless sensor nodes and used an energy-efficient data transmission schedule to obtain real-time data for pH, conductivity, calcium, temperature, fluoride, and dissolved oxygen. The author's system architecture includes a Libelium Waspmote that uses communication modules such as 3G/4G, General Packet Radio Service, long-range 802.12.4/ZigBee, and Wideband Code Division Multiple Access connectivity to transmit data to the cloud. Their sensors detected conductivity levels between 196-225 S/cm, temperatures of 35-36 ℃, calcium levels between 0.16-3.5 mg/L, maximum DO levels of 8 mg/L, and fluoride levels between 1.24-1.9 mg/L. A low-cost Arduino-based sonde was designed in the marine environment that used the Arduino Mega 2560 Mega and Arduino

Uno platforms for two design configurations (Lockridge et al., 2016) The first design was for a Lagrangian style drifter that Lockridge et al. deployed for 55 hours to measure salinity and temperature using Atlas Scientific K 1.0 Conductivity Probe and the Atlas Scientific ENV-TMP temperature probe. The data collected by the drifter was compared to values from the Dauphin Island Sea Lab Weather Station YSI 6600 sonde. The RMS error was 1.35 ppt for the salinity and was 0.154 °C for the temperature measurements. Salinity and temperature regressions were also performed and found to be highly correlated with $R^2 = 0.96$ for salinity and $R^2 = 0.99$ for temperature. The first data logger demonstrated consistently higher salinity results than the second, whereas temperature tracked similarly for both data loggers over the entire deployment. The system contributed significantly to verifying sensor performance for harsh aquatic environments. However, the work only stored data locally as .csv files and did not transmit data wirelessly or in real-time.

For IoT-enabled systems, some of the most straightforward system configurations are proposed by Vijayakumar & Ramya (2013) and Pasika & Gandla (2020). Work by Vijayakumar presents a low-cost system for developing a real-time water quality monitoring system in the IoT environment. The core controller used is the Raspberry Pi B+, and the system has temperature, turbidity, pH, conductivity, and dissolved oxygen sensors. The Raspberry Pi runs on a LINUX kernel and uses an external USR-WIFI232-X-V4.4 module to transfer the data to the internet and is visible on the ioBridge Server. Pasika et al. propose a system consisting of four sensors with the Arduino Mega microcontroller. Their approach is designed in Embedded-C and accomplishes data transmission using the ESP8266 Wi-Fi module. Authorized users can access the data using a User ID and password to log into a ThingSpeak server. The information is gathered, stored, analyzed, and transmitted in real-time.

Similar systems to those mentioned previously build upon similar design principles but include additional features incorporated into the software designs, such as power-saving capabilities, user alerts, and contamination detection.

Power saving capabilities are primarily emphasized for more rugged environments where connectivity issues and power loss can be concerns. One such area is Malawi, the site for a proposed integrated sensor network (Zennaro et al., n.d.) The solution was to develop a low-power gateway node that reduces energy consumption using a wake-up mechanism that triggers waking and sleeping modes. A gateway solution built upon the ALIX2 embedded Linux board provided a way to interconnect different networks to address connectivity issues. Parra et al. (2018) also addressed power-saving algorithms for their fish farm monitoring wireless sensor network, which was achieved by only sending data if the difference between predesignated reference and threshold values were exceeded. Thus, reducing the amount of data moved to the external web server employed in their system and saving computing power.

Threshold values can also be employed for systems alerting users of parameter exceedance of WHO or EPA guidelines. This is the case in the IoT-enabled system developed by Geetha & Gouthami (2016) which measures conductivity, turbidity, water level, and pH. The hardware design consists of a TI CC3200 single-chip microcontroller with an in-built WIFI module, and ARM Cortex can connect to the nearest Wi-Fi hotspot. The data collected by the sensors is sent to the cloud, and if the values exceed threshold limits obtained from WHO, then an alert is sent to the mobile application developed as part of the proposed system. The programming for the software design that shows the real-time updates was done using ENERGIA IDE, and data are stored in the Ubidots cloud, which includes a real-time dashboard to analyze data, control devices, and shares the data through public links. Lambrou et al. (2014)

further developed the event detection capabilities of wireless sensor networks by designing and developing a low-cost network embedded system consisting of a central measurement node, a control node, and a notification node. The system contained sensors to measure temperature, turbidity, electrical conductivity, and pH. These three subsystems served to collect water quality measurements from the sensors, implemented contamination event detection algorithms, and stored the measured data in a local database, visualized the data in the form of charts, and sent email/SMS alerts for contamination events. The system architecture utilizes a PIC MCU (Programmable Interface Controllers Microcontroller Unit), an ARM processor, and a ZigBee RF transceiver. The data is posted to the web using the Pachube IoT platform. For the contamination event detection algorithms, the first system was denoted as the Vector Distance Algorithm with a risk indicator function estimated on the Euclidean distance between the normalized sensor signal vector and the normalized control signal vector of clean water. The second was the polygon area algorithm which calculated a separate risk indicator function estimated on the ratio of the polygon area formed by the vector components of each sensor on a two-dimensional spider graph. To validate the event detection algorithms, intentional contamination was performed using E. Coli bacteria and Arsenic, which was added to potable water at various concentrations at discrete time intervals. For E. Coli, the Vector Distance Algorithm and the polygon area algorithm both missed the detection of $5\ x10^{-2}\ CFU/mL$. For Arsenic, the sensors did not respond for concentrations lower than $25\ \mu g/L$, but both pH and ORP sensors responded at higher concentrations. Overall, the PAA had better performance for both contaminants due to fewer false alarms.

It is with these developed wireless sensor networks in mind that the prototype in this work has been uniquely designed to combine, improve upon, and fill in gaps from the existing technologies and methods for applications specific to point-of-use water filtration applications.

Water Quality Correlations

After investigating the current hardware and software utilized for wireless sensor networks, a key research question to be addressed was to determine the sensors and algorithms that were best suited for smart point-of-use water quality monitoring. Therefore, some of the existing relationships between various sensor parameters and water quality concerns were reviewed to better guide the sensor selection process. Here, we present some of the findings on what water quality information can be related to pH, ORP, DO, and EC.

For acute effects from biological contaminants, it is imperative that sufficient disinfection levels are maintained. In drinking water, this is primarily achieved with Chlorine and Chloramine, which previous research has attempted to link to ORP measurements. Experiments have been conducted that investigate the ability of ORP to be a predictor for kill level of organisms including total coliform, E. Coli, and enterococci by measuring the ORP in conjunction with Chlorine dose for wastewater (Bergendahl & Stevens, 2005). It was found that the ORP increased with an increase in chlorine added, total chlorine, and free chlorine. However, the regression slopes were relatively low indicating low sensitivity of ORP measurements as a function of each chlorine species. However, the authors highlight that a large change in measured ORP occurs near zero chlorine residual and commented on the effectiveness of controlling chlorine with ORP measurements at low chlorine concentrations, but as the free chlorine concentration increases beyond 1 mg/L the sensitivity of the measurement is reduced.

A pilot-scale application of using ORP and pH to develop a control strategy for chlorination of wastewater was achieved by using significant points occurring on the pH and ORP profiles during chlorination titrations (Kim et al., 2006). The results showed that ORP increased from $300\ mV$ to $400\ mV$ when Chlorine solutions were first added, then the ORP profile flattened as NH_2Cl dominated the redox state of the water. At the same time, the pH profile increased due to the hydroxyl ion formed from $NaOCl$. As more chlorine was added, a second increase in ORP was observed. From this point, Monochloramine is oxidized to Dichloramine, which has a higher redox potential and produces H^+ leading to a lower solution pH. Once all the Dichloramine was oxidized, free chlorine became available, and the third increase in ORP was observed, indicating the chlorine breakpoint after which viable counting of microorganisms was not observed. The control system implemented could continuously detect the breakpoint and determine the correct chlorine does for inactivating microorganisms in the water. The proposed system ultimately can easily detect the shifting point where dominating chemical species change by monitoring points along with the ORP and pH profiles during chlorination to achieve proper disinfection.

ORP measurements have also been explored for monitoring and controlling water disinfection for the produce washing industry (Suslow, n.d.). ORP ranges between $600\ mV$ and $700\ mV$ show that free-floating decay and spoilage bacteria, as well as pathogenic bacteria such as E. coli or Salmonella species, are killed within 30 seconds. In relation to the pH sensors, lowering the pH raises the percentage of $HOCl$, and ORP increases to reflect this shift in oxidative potential. Recent research in commercial and model postharvest water systems has shown that, if necessary, ORP criteria can be relied on to determine microbial kill potential across a broad range of water quality. However, it is important to note the limitation of ORP and

pH because the effect of pH on chlorine speciation, one must use caution in not having a false sense of adequate disinfection rates at high pH's. In general, a ten-fold increase in total or free chlorine concentrations does not result in a corresponding proportional increase in ORP millivolts. Their results showed that good water quality likely results in measurements of $650\ mV$ to $700\ mV$ ORP if the water pH is 6.5 to 7. Lowering the pH to 6.0 raises the ORP as more hypochlorous acid becomes available. Raising the pH to 8.0 lowers the ORP value, as more hypochlorite ions are present. Maintaining constant pH but adding more chlorine raises the ORP to a plateau of about $950\ mV$. Finally, Myron L Company describes their correlation for predicting free available chlorine using ORP and pH levels. The correlation was obtained from a series of experiments where an exact amount of chlorine in the form of laboratory-grade bleach was added to DI water in a closed system and measuring pH and ORP to create calibration curves for their Myron L Ultrameter II 6PFCE Water Quality Meter. From some of the listed works, it is evidenced that ORP can be used as an effective indicator for disinfection levels in water and has the potential to protect users from potential biological contamination using real-time measurements of ORP in a wireless sensor network.

Other correlations between the variables measured in the wireless sensor network proposed in this work and water quality have been discovered utilizing statistical data analysis methods and machine learning techniques. In 2014, Zhang et al. focused on robust online clustering (ROC) and modified pixel-based adaptive segmentation (MoPBAS) and concluded that the MoPBAS method was suitable for detecting anomalous sensor readings and event clustering from salinity and turbidity measurements, which can assist in addressing root environmental causes and significance levels of disturbance events. Forough et al. (2019) used an alternative machine learning technique known as the support vector machine (SVM) model to

predict a water quality index (WQI) using pH, DO, TDS, temperature, Nitrate, phosphate, BOD, turbidity, and fecal coliform data obtained from water samples collected over 11 months. Their SVM model, developed in MATLAB, successfully explained 87% of the variability in total WQI, with Nitrate being the most important attribute influencing the WQI as calculated from the sensitivity ratio. Machine learning techniques have also been combined with regression models, as demonstrated by Saetta et al. (2021) who developed a sensor platform to predict chlorine residuals in university buildings. DO, pH, EC, ORP, and free chlorine data were collected, and two models were developed. The first is a linear regression model, and the second is a gradient boosting machine (GBM) model. In R statistical software, the "ggpairs" function and "leaps" function to create linear regression models which were unable to predict free Chlorine levels, however, the GBM models had lower RMSE values than their multivariate linear regression counterparts, and the t-tests showed that there was no significant difference in the predicted vs. actual data, showing the sensors used in this study could be used to predict chlorine by using the GBM model. Despite the difficulty in predicting water quality using linear regression several studies (Fathi et al., 2018; Leventeli & Yalcin, 2021) used multivariate and non-linear regression techniques to estimate heavy metal content, nutrients, and bacteria concentration, as well as WQI's in various water sources with R^2 values as high as 0.90. Factors including EC, pH, turbidity were among some of the most important in influencing the overall predictions.

The relationships explored in the work reviewed serve as a guiding framework for the proposed prototype and give insight into expected sensor behaviors.

Method

Various organizations, including the World Health Organization, The European Union, and the United States Environmental Protection Agency, set drinking water quality standards to

ensure that drinking water is clean and safe for consumers (CDC, 2021). For the prototype system proposed, the parameters to be monitored from the sensor package were based on their relationships to chemical and biological contamination, with particular emphasis on selecting the minimum amount of key sensor measurements with the greatest predictive water quality capabilities. The selection process for the sensors included in this prototype is detailed in the following section

Sensor Selection

Lab grade sensors that use various electrochemical mechanisms were selected from the manufacturer Atlas Scientific. Primary factors that were considered when selecting the sensors from off-the-shelf manufacturers to be used for the prototype included sensitivity, selectivity, stability, lifetime, response time, pressure tolerance, cost, and size. In addition, the robustness, ease of calibration, compact size, and commercially available plumbing components contributed to the overall selection process for the sensor probes to be included in the prototype. The complete sensor package includes sensor probes for pH, EC, DO, ORP, and temperature. Table 1 describes specifications for each of the selected sensors.

Table 1:*Sensor Specifications*

Probe Type	pH	ORP	DO	EC
Range	0-14	+/- 2000 mV	0-100 mg/L	0.07-50,000 uS/cm
Resolution	+/- 0.001	-	-	-
Accuracy	+/- 0.002	+/- 1 mV	+/- 0.05 mg/L	+/- 2%
Response Time	95% in 1s	95% in 1s	~ 0.3 mg/L/ per sec	90% in 1s
Temperature Range	-5-99 ºC	1-99 ºC	1-60 ºC	1-110 ºC
Max Pressure	100 PSI	100 PSI	500 PSI	500 PSI
Max Depth	70m (230 ft)	70m (230 ft)	352m (1,157 ft)	352m
Internal Temperature Sensor	No	No	No	No
Time Before Recalibration	~ 1 Year	~ 1 Year	~1 Year	~10 Years
Life Expectancy	~2.5 Years	~ 2 Years	~ 4 Years	~10 Years

As evidenced by some of the related work on wireless sensors, these can be used as indicators for a wide range of water quality parameters. The sensors all operate on different electrochemical principles that convert raw voltages into digital values. pH is a measure of how acidic or basic water is and is important in water quality because it determines the solubility and biological availability of chemical constituents, nutrients, and heavy metals. Metals tend to be more toxic at lower pH because they are more soluble. The pH probe measures the hydrogen ion activity in liquids and has a glass membrane where hydrogen ions in the liquid diffuse onto the outer layer of the glass while larger ions remain in solution. The difference of concentration of hydrogen ions on the outside of the probe vs. inside the probe creates a measurable current that is proportional to the concentration of hydrogen ions in the liquid. The probe includes an internal double junction, an EXR glass tip, and a body of extruded epoxy, making the probe suitable for

pH measurements of high purity waters and capable of resisting strong acids and bases for contaminated waters.

Electrical conductivity was another parameter to be measured. Conductivity is the measure of a water's ability to pass an electrical current. Substances that conduct electrical current include dissolved salts and other inorganic chemicals. Most waters will have a relatively constant range for conductivity; therefore, significant changes can be good indicators of the pollution of an aquatic resource. Waters with elevated conductivity may have other impaired or altered indicators as well. Inside the conductivity probe, two electrodes are positioned opposite each other. An AC voltage is applied to the electrodes, which results in the cations moving to the negatively charged electrode while the anions move to the positively charged electrode.

The next sensor parameter selected was dissolved oxygen. The Atlas Scientific DO probe consists of a PTFE membrane, an anode bathed in an electrolyte, and a cathode. The operating principle is based on the oxygen molecules diffusing through the membrane of the probe at a constant rate. After crossing the membrane, the oxygen molecules then reach the cathode, where they are reduced, and a small voltage is produced, which is read by an analog to digital converter. Dissolved oxygen is important in drinking water because high DO levels can damage components and systems that are used in drinking water treatment and distribution (Jung et al., 2009) Namely, high DO levels can contribute to corrosion in pipes, and too low of levels can create issues with the taste of water.

ORP is a measure of the oxidation-reduction potential where oxidation is the loss of electrons and reduction is the gain of electrons. An ORP probe measures electron activity in a liquid and shows the strength at which electrons are transferred to or from a substance in a liquid. The ORP probe selected contains a platinum tip and a 4 molar KCl reference solution. ORP was

selected because it, combined with pH and temperature, can be used to estimate free chlorine concentrations, which is an indicator of the presence or absence of disease-causing bacteria and viruses, these are typically the cause of most acute symptoms of waterborne disease or illness. ORP was also selected because it can be used in combination with pH and metal concentrations to plot the equilibrium potential of electrochemical reactions. This is useful in predicting the corrosion risk and speciation of various chemical constituents in aqueous solutions and is incorporated into the back-end software package.

Sensor Calibration

To prepare the sensors for data collection of tap water samples, they were first calibrated. This was achieved by running the Atlas Scientific Raspberry Pi Sample Code, an open-source code provided by the manufacturers, on a Raspberry Pi microprocessor. The pH sensor was calibrated in pH 4.01, pH 7.00, and pH 10.01 buffer solutions from Hach. The remaining sensors were calibrated using the accompanying Atlas Scientific calibration solutions. The conductivity sensor underwent a 2-point calibration with $12{,}880\ \mu S$ and $150{,}000\ \mu S$ solutions, DO also had a two-point calibration in dry air and a zero dissolved oxygen solution. Lastly, ORP and temperature were calibrated using $225\ mV$ calibration solution and $100\ ℃$ boiling water, respectively. After calibration, the sensor performance was verified by comparing the values of the five sensors against values from a Myron ULTRAMETER II™ 6PFC[E]

System Architecture

To complete the referenced calibration procedures, the sensors were integrated as part of the embedded system for continuous water quality monitoring of the 5 main parameters. The system can primarily be split into hardware and software components.

Hardware Components

For the hardware, the central measurement system consists of a microprocessor, the five sensor probes, and two expansion boards outfitted with EZO embedded circuits designed for each sensor probe by the manufacturers Atlas Scientific.

The microprocessor used in this system is the Raspberry Pi Model 4 B. It acts as the gateway to collect the information from the sensors and transfer the collected data to the Git repository via a wireless network. For point-of-use applications, the device chosen for the microprocessor should have high storage capabilities, flexible connectivity, be low-cost, and have ample computing power to run any necessary programs. The Raspberry Pi meets these

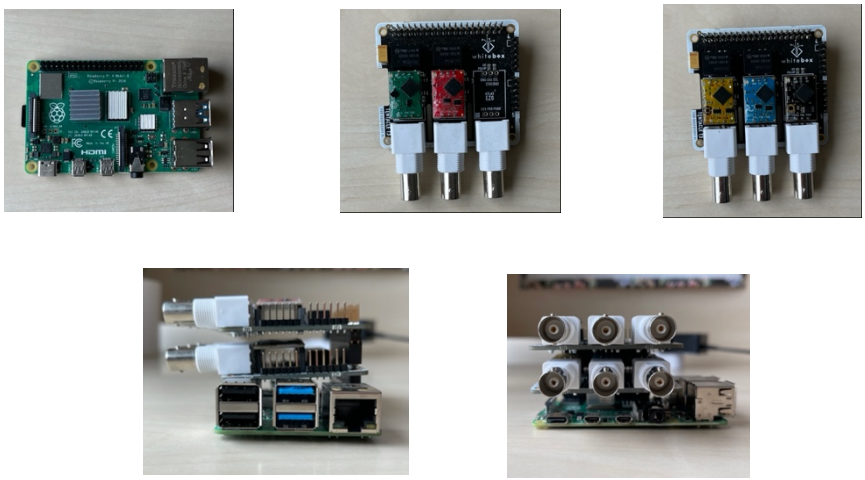

Figure 1: System Baseboard and Carrier Boards

requirements with its powerful 1.5 GHz 64-bit quad-core ARM Cortex-A27 processor, onboard 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet, and 2-8 GB of RAM. The Raspberry Pi 4 Model B also has two USB 2.0 ports, two USB 3.0 ports, and a standard 40-pin GPIO header. The GPIO pins connect to the WhiteBox Labs carrier boards. The carrier boards are stackable and contain 6 slots for the Atlas Scientific EZO circuits. These features eliminate the need for

multiplexing, wiring, and breadboards. Up to six sensors can be connected at once for data collection with these two carrier boards.

The carrier boards connect directly to the Raspberry Pi pins, this allows for easy establishment of serial communication using I2C protocol between the microprocessor and the circuits. The system also contains auxiliary hardware components, including a keyboard, mouse, and a 7 in. LCD screen, which allows the operator to run the data acquisition python scripts.

Figures 1(a-e) show the various individual components of the system. Figure 2 shows the completely assembled system, including the auxiliary hardware.

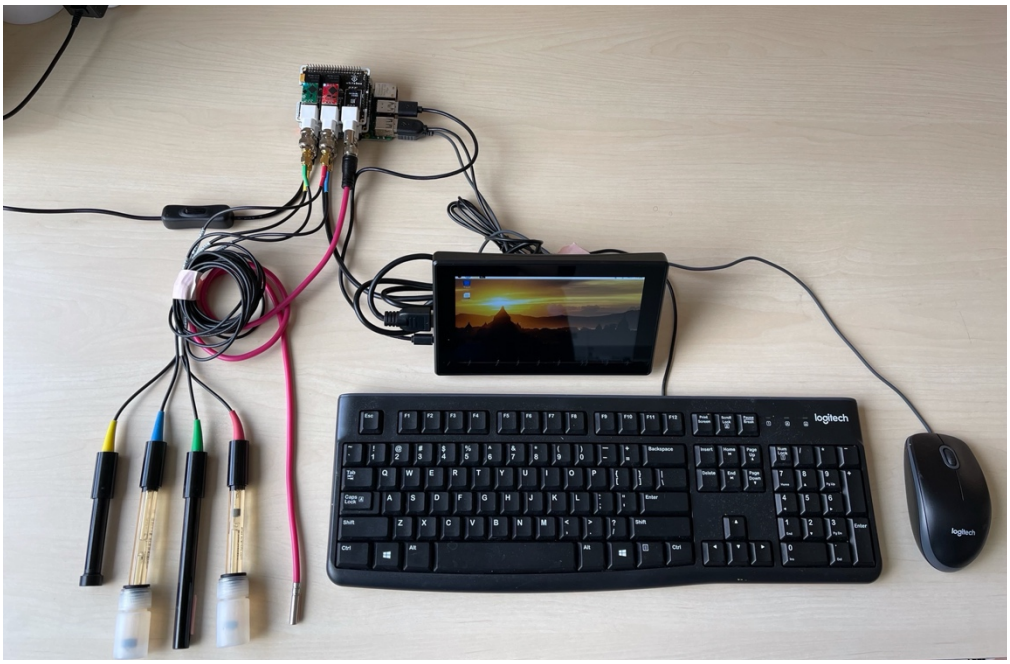

Figure 2: Assembled Wireless Sensor Network

Software Components

The code for the prototype was written in python integrated development environment software. Source code from Atlas Scientific was the starting point for the data collection, and modifications were made for customized formatting and real-time data transmission. Once the data acquisition code is initialized by an operator, DO, pH, EC, temperature, and ORP data were

received from the sensor in 1s intervals. The data were temperature compensated, parsed into integers, assigned units, positioned into arrays, and displayed in the terminal window during data collection. To save the data, a new file was created, and the collected data was written and saved locally onto the micro-SD card. The software program also automatically pushed the data collected from the microprocessor to a cloud-based GitHub repository via a wireless network. The system connects to the GitHub repository using the following steps:

1. Connect to the sensor access point using the LCD screen and open the terminal window
2. Navigate to the Atlas Scientific Data Acquisition directory
3. Run the ./start command to begin data collection
4. Enter the required user inputs, including
    a. Login Credentials
    b. Select yes or no option for real-time data plotting
    c. Add optional comments to describe run/operating conditions
5. Run ctrl-c to stop data collection
6. Confirm if the user would like to keep the data set by inputting yes or no
7. Re-enter login credential to push data to GitHub account

Appendix A contains the source code used for the data acquisition process described above. demonstration of the terminal window during data collection from a tap water sample was captured in Figure 3

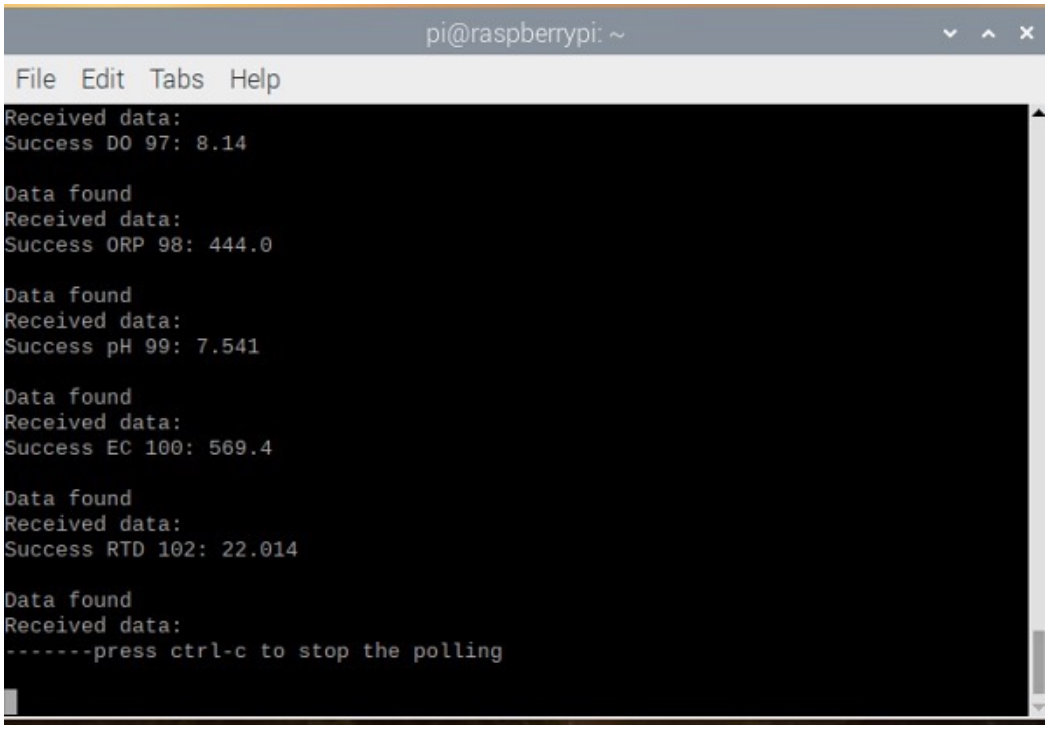

Figure 3: Terminal Window for Tap Water Data Collection

After following the listed steps, the collected data will be accessible to any collaborators with access to the GitHub account. Features on the repository show each data collection run as separate .csv files that can be easily exported to other data processing software such as excel, R, or SPSS.

Besides the real-time data collection, an additional program was developed for back-end data processing using the ORP and pH sensor data. The back-end algorithms consist of Pourbaix diagrams developed for copper, zinc, iron, and lead at standard temperatures and pressure. These were created using the Nernst equations for redox reactions and acid-base reactions transcribed in Marcel Pourbaix's Atlas of Electrochemical Equilibria in Aqueous Solutions (Pourbaix, 1974)

Pourbaix diagrams, are electrochemical graphs that show possible thermodynamically stable phases for aqueous systems. Using Pourbaix diagrams one can predict the equilibrium states of all the possible reactions between an element, its ions and its solid and gaseous compounds in the presence of water. To construct a Pourbaix diagram a standard chemical

potential and ion activities must be assumed for the substances reacting. These diagrams can be read similarly to a standard phase diagram with electrical potential and pH as the axes. The lines of a Pourbaix diagram are developed using the Nernst Equation and show the equilibrium conditions for the species on each side of that line where each species on either side is said to predominate. For a reversible redox reaction with the following equilibrium equation

$$aA + bB \leftrightarrow cC + dD$$

With and equilibrium constant

$$K = \frac{[C]^c[D]^d}{[A]^a[B]^b}$$

The Nernst Equation may be expressed as

$$E_H = E^0 - \frac{RT}{zF} lnK$$

Where E^0 is the standard potential, R is the universal gas constant, T is temperature in Kelvin, F is Faraday's Constant, z is the ion charge, and K is the equilibrium constant. Substituting the equilibrium constant gives,

$$E_H = E^0 - \frac{RT}{zF} ln \frac{[C]^c[D]^d}{[A]^a[B]^b}$$

This equation is sometimes simplified to

$$E_H = E^0 - \frac{V_T \lambda}{z} log \frac{[C]^c[D]^d}{[A]^a[B]^b}$$

Where

$$V_T = \frac{RT}{F} \approx 0.02569\, Volts$$

Is the thermal voltage or the "Nernst slope" at standard temperature. The equation can further be simplified using

$$\lambda = \ln(10)$$

Thus, the equation can be numerically expressed as

$$E_H = E^0 - \frac{0.05916}{z} log \frac{[C]^c[D]^d}{[A]^a[B]^b}$$

Using these equilibrium formulae, one can construct diagrams for various metals of concern in aqueous system. To draw the position of the lines with the Nernst equation, the activity of the chemical species at equilibrium must be defined, for many soluble species the concentrations are assumed 10^{-6} M, this convention was followed for the software package developed. Changes in temperature and concentration of solvated ions will affect the equilibrium lines as dictated by the Nernst Equation. The diagrams presented in this work are for metals that are common contaminants in premise plumbing and include lead, copper, iron, and zinc. To establish the diagrams, it must first be noted that there are three types of lines in Pourbaix diagrams: Vertical, horizontal, and sloped.

Vertical lines represent reactions where no electrons are exchanged and are acid-base reactions. This will create a boundary line that is vertical at a designated pH value and the reaction involves only protonation/deprotonation. Using the reaction equation

$$aA + bB + hH^+ \leftrightarrow cC + dD$$

And the energy balance

$$\Delta G^0 = -RTln(K)$$

Where the equilibrium constant K takes the form

$$K = \frac{[C]^c[D]^d}{[A]^a[B]^b[H^+]^h}$$

Therefore, the energy balance can be written as

$$\Delta G^0 = -RTln\left(\frac{[C]^c[D]^d}{[A]^a[B]^b[H^+]^h}\right)$$

Or, in base-10 logarithms

$$\Delta G^0 = -RT\lambda\left(log\left(\frac{[C]^c[D]^d}{[A]^a[B]^b}\right) + hpH\right)$$

Which may be solved for a particular value of pH. For the establishment of Lead the vertical lines distinguish between the limits of the domains of predominance for the dissolved substances and the relative stability of the solid substances.

Horizontal lines are created from equilibrium equations for reactions that do not involve H^+ or OH^- thus the boundary line is independent of pH and the reaction equation may be written as

$$aA + bB + ze^- \leftrightarrow cC + dD$$

The new energy balance can be expressed as

$$\Delta G^0 = -RTln\left(\frac{[C]^c[D]^d}{[A]^a[B]^b}\right)$$

And from the electrode potential

$$\Delta G = -zFE$$

The Nernst Equation for the horizontal lines takes the form

$$E_h = E^0 - \frac{V_T}{z}\ln\left(\frac{[C]^c[D]^d}{[A]^a[B]^b}\right)$$

Or for the base-10 logarithms form

$$E_h = E^0 - \frac{V_T\lambda}{z}\log\left(\frac{[C]^c[D]^d}{[A]^a[B]^b}\right)$$

These may be solved for horizontal lines at specific voltages dependent on the ratio of concentrations for the chemical constituents of each diagram.

Lastly, there are the sloped boundary lines which involve both electrons and H^+ ions.

$$aA + bB + hH^+ + ze^- \leftrightarrow cC + dD$$

The Nernst equation become in base-10 logarithm form

$$E_h = E^0 - \frac{V_T \lambda}{z} \log\left(\frac{[C]^c[D]^d}{[A]^a[B]^b} + hpH\right)$$

Where the h value is the slope of the line. These were then used as the basis for a python script which uses Matplotlib visualization tool to generate the diagram. The pH and ORP value from the sensors are then used to place a point on the Pourbaix diagram. That point is then evaluated using different logic operators and the python Shapely package to determine if the point falls within the immunity, corrosion, or passivation region. The code also returns the predominate species for the aqueous system and their state. This feature was included due to metal contamination from premise plumbing being a primary concern for point of use applications. The code for the diagrams is included in Appendix B and contains all the solved Nernst equations for the lines needed to develop the diagrams for lead, copper, iron, and zinc.

Experimental Design

With the software and hardware components successfully operating the next step in the research process was to experimentally test the sensors to ensure they performed well in drinking water applications and determine if the prototype system could be used for predictive modelling or event detection. Three primary experimental tasks were involved.

Experimental Phase One: Sensor Validation

Reliable sensor measurements are essential for smooth operation of the proposed prototype. If a sensor fails to provide accurate measurements false ideas of the water quality are perpetuated creating potential health risks or leading to suboptimal operation. Thus the, first task for testing the system was to ensure the sensors returned accurate and reasonable results for tap water samples. The procedure for preparing the prototype for sample collection so it could be tested in drinking water is detailed in Appendix C. The practice of sensor validation approach

consists of using multiple sensors for the same parameter measurement. This technique can determine if a sensor is faulty or if high deviations between multiple sensors exists. This technique was applied by first by measuring the pH, EC, DO, temperature, and ORP for a given sample with the Atlas Scientific sensor probes. The pH, EC, temperature, and ORP of the same sample were then measured using a laboratory grade instrument to compare the results between the two devices. The Myron Ultrameter was selected as the comparative instrument due to its streamlined and accurate functionality.

A total of 34 tap water samples for testing were collected from a laboratory tap water faucet at the University of California Los Angeles between the hours of 10 AM to 4 PM from the months of August 2021 through October 2021. The samples were collected in 1000mL volumes in a graduated cylinder. Approximately 100mL of the sample was immediately tested using the Myron Ultrameter for pH, ORP, EC, and temperature. This process consisted of 1) cleaning the cell cup thoroughly with MilliQ water 2) turning on the Myron Ultrameter 3) rinsing the sample cell 3 times with the sample to be tested 4) refilling the sample cell with additional sample 5) pressing the desired measurement key, and 6) recording the values. The remaining ~900mL volume of the sample was transferred to a 1000mL beaker and placed on a hot plate with a magnetic stirrer. The sample was stirred at approximately 750 Rpms to mimic the flow of water through an inline pipe system and provide sufficient flow for the DO sensor. The clean and dry sensor probes were then inserted into the beaker, ensuring that the sensors did not touch the stir bar and had minimal contact with the walls of the 1000mL beaker. For each sensor run the hotplate was maintained at 25℃; however, some small temperature variations were observed during testing due to factors such as varying ambient room temperatures and temperature changes in the building pipes throughout the day. Data collection was then initiated using the

steps described in the previous software section. The sensors collected one data point every second continuously for 10 minutes to allow sufficient sensors stabilization time. The values for the 10 minute run were then downloaded from the GitHub into an Excel file. The mean value for each sensor was calculated for that sample and recorded in an Excel file with the corresponding Myron Ultrameter value for the same sample. From the remaining ~900mL of sample, another 100 mL was set aside and refrigerated for additional laboratory analytical testing.

Once collected this data set was analyzed using two statistical tests recommended for comparing two instruments with continuous data as outlined in NIST Technical Note 2106. In that document the null hypothesis states:

$$H_0: All\ instruments\ perform\ equally\ well$$

While the alternative states:

$$H_A: Not\ all\ instruments\ perform\ equally\ well$$

For instrument comparison, the p-value can usually be interpreted as the probability of measuring a disparity as great or greater than that seen, under the assumption that the instruments are truly equivalent. If the p-value is smaller than $\alpha = 0.05$ then the null hypothesis is rejected by the test.

To obtain an appropriate p-value, first the variances between instruments were compared for each measured parameter using an. F-test

$$F = \frac{s_1^2}{s_2^2}$$

Where,

$$s^2 = \frac{\Sigma(x - \overline{x})^2}{n - 1}$$

And

$$s^2 = variance$$

$$x = values\ in\ data\ set$$

$$\overline{x} = mean\ of\ the\ data$$

$$n = total\ number\ of\ values$$

After determining if the variances were equal or not, the appropriate test was applied. For equal variance test

$$T = \frac{\overline{x} - \overline{y}}{s_p\sqrt{\frac{1}{m} + \frac{1}{n}}}$$

Or in the case of unequal variance

$$T = \frac{\overline{x} - \overline{y}}{\sqrt{\frac{s_x^2}{m} + \frac{s_y^2}{n}}}$$

Where,

$$\overline{x} = the\ mean\ of\ the\ measurements\ from\ instrument\ 1$$

$$\overline{y} = the\ mean\ of\ the\ measurements\ for\ instrument\ 2$$

$$s_x^2 = the\ sample\ variance\ for\ instrument\ 1$$

$$s_y^2 = the\ sample\ variance\ for\ instrument\ 2$$

The result of this comparison was used to determine if the probes selected for the prototype gave reasonably accurate results compared to a multiparameter benchtop probe.

Experimental Phase Two: Predictive Model Development

The subsequent phase of the experimental process was the testing of each of the 34 tap water samples for EPA primary and secondary drinking water standards. The purpose of this was to attempt to develop models between the sensor values recorded in phase one and correlate them to other important drinking water values not measured directly by the sensors but were hypothesized to have some correlation due to physical or chemical phenomena. Turbidity and dissolved organic carbon were chosen because of their potential to be correlated with dissolved oxygen since these parameters are often indicators of microbial activity, which consumes oxygen in water. Various transitions metals were also chosen because metals would presumably contribute to the overall electrical conductivity of a sample, plus ion specific sensors are extremely cost prohibitive so there are economic benefits to attempting to model their concentrations in water. Free chlorine was chosen because it relates highly to the pH and ORP sensor values as discussed in the related work section. Lastly, color was chosen because it is one of the main visual indicators of poor water quality, and therefore a major consumer concern, although it was not expected that any of the sensor values could be used to predict color as they do not feature any optical measurement features. Table 2 lists the parameters tested, the standard method performed for each, and the instrument utilized

Table 2:*Primary and Secondary Drinking Water Parameters Tested*

Parameter	Method	Instrument
Turbidity	USEPA Method 180.1	Hach 2100P Handheld Turbidity Meter
Color	ASTM D1209	Hach 2100AN Turbidimeter
Free Chlorine	Hach Method 8021	Beckman DU-530 UV-VIS spectrophotometer
Dissolved Organic Carbon	SM 5310B	Simadszu TOC-L series
Zinc	USEPA Method 6010D	Avio 220 Max ICP Optical Emission Spectroscopy
Iron	USEPA Method 6010D	Avio 220 Max ICP Optical Emission Spectroscopy
Copper	USEPA Method 6010D	Avio 220 Max ICP Optical Emission Spectroscopy
Lead	USEPA Method 6010D	Avio 220 Max ICP Optical Emission Spectroscopy
Manganese	USEPA Method 6010D	Avio 220 Max ICP Optical Emission Spectroscopy

The results obtained from the additional laboratory testing of the samples were matched the sensor values obtained in phase one and added to the database.

The statistical technique chosen for the design of the models was multiple linear regression. This method involves using several explanatory variables to predict the outcome of a response variable. This technique can be used to determine how strong the relationship is between two or more independent variables and one dependent variable. The technique allows one to obtain a predicted value for specific variables. To perform multiple regression several assumptions must be met.

Four principal assumptions were tested to suit the suitability of the data for multivariate regression. 1) there must be a linear relationship between the outcome variable and the independent variable 2) the residuals are normally distributed 3) there is no multicollinearity, that is the independent variables are not highly correlated with each other, and 4) homoscedasticity must be satisfied meaning the variance of error terms are similar across the values of the independent variables.

A general rule of thumb for multilinear regression is approximately 20 cases per independent variable in the analysis. For testing the linearity scatterplots may be constructed to ensure the data display linear behavior, the data collected over the course of this experiment were screened graphically for any behavior suggesting non-linearity such as exponential, logarithmic or polynomial. To further confirm the linearity Pearson's correlation was calculated to measure the linearity between the data. In statistics Pearson's correlation coefficient is defined as

$$r = \frac{\Sigma(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\Sigma(x_i - \overline{x})^2 \Sigma(y_i - \overline{y})^2}}$$

Where,

$$r = correlation\ coefficient$$

$$x_i = values\ of\ the\ x - variable\ in\ a\ sample$$

$$\overline{x} = \ mean\ of\ the\ values\ of\ the\ x - variable$$

$$y_i = values\ of\ the\ y - variable\ in\ a\ sample$$

$$\overline{y} = mean\ of\ the\ values\ of\ the\ y - variable$$

Multicollinearity was also calculated by creating a correlation matrix using Pearson's correlation, via the same method.

The next assumption is the normality of the data, this was tested by calculating and sorting the standard residuals. The frequency values were then calculated, and histograms were generated to observe the distribution. To find the residuals let

$$r = x - x_0$$

Where,

$$r = residual$$

$$x = measured\ variable$$

$$x_0 = predicted\ variable$$

From this, the standard residuals were obtained via software in the Excel Data Analysis Toolpak. The final assumption is for homoscedasticity, this can be tested for using a statistical test known as the Breusch-Pagan test; it is its own linear regression model. It tests whether the variance of errors from a regression depends on the values of the independent variables. It is a chi-squared test and if the test has a p-value below the threshold of $p < 0.05$ then the null hypothesis of homoskedasticity is rejected. The following equation was used for the chi-squared value

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

After which the excel function $= \text{CHISQ. DIST. RT}(\chi^2, \text{df regression})$ was used to obtain the p-value for each set of regressions. The data was tested to assure it met all the necessary assumptions and multivariate regression was then performed. Because each predicted variable required its own model, nine separate regression analyses were performed for the parameters listed in Table 2. Statistical software typically use a curve estimation procedure, which was used to produce regression statistics for each of the desired dependent variables. To develop each of the equations it can be said that each dependent variable Y depends on X. For multiple regression

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_\rho X_\rho + \sigma(Y), sd(Y) = \sigma$$

Where

$$\beta_0 = intercept$$

$$\beta_1 \ldots \beta_\rho = regression\ coefficients$$

$$\sigma = residual\ standard\ deviation$$

For this project, there are $\beta_{1,2,3,4,5}$ which are the coefficients obtained from each sensor value for a given drinking water sample. Given that there are 5 sensors, to obtain each prediction equation

a 5x5 system of linear equations was solved. Once the coefficients were determined for each outcome variable the predicted values were compared to the actual values. A regression model deemed to be a good fit ideally results in predicted values close to the observed values. Two statistical tests are appropriate for evaluating the differences between predicted and observed values. The first was the R^2 value or in the case of multiple linear regression the adjusted R^2 value. The adjusted R^2value incorporates the degrees of freedom of the model and explains the proportions of total variance in the model. The adjusted R^2 was also obtained using software featured in Excel's data analysis Toolpak, however the R^2 may be calculated directly using Pearson's correlation coefficient where,

$$R^2 = (r)^2$$

From this,

$$R^2_{Adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where,

$$R^2 = sample\ R - square$$

$$p = number\ of\ predictors$$

$$N = total\ sample\ size$$

In addition to the adjusted R^2 value, the root mean square error (RMSE) was also calculated. This statistic indicates the absolute fit of the model to the data and is the most important criterion for fit if the main purpose of the model is prediction. The RMSE can be expressed as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \widehat{x_i})^2}{N}}$$

Where,

$$N = total\ sample\ size$$

$$x_i = observed\ value$$

$$\widehat{x_i} = predicted\ value$$

Using the above experimental and data analysis techniques the predictive capabilities of the system, and the contribution of each sensor to additional parameters of concern can allow for the utilization of the prototype to be expanded beyond real time monitoring, increasing the systems utility for ensuring clean drinking water.

Experimental Phase Three: Contamination Detection

The final set of experiments were the collection of data for contamination events for future testing of outlier detection algorithms and event detection systems. For these experiments, the Atlas Scientific sensor probes were inserted into a 2L beaker, with water from the laboratory faucet running into the beaker at 2.67 $\frac{L}{min}$ through a MasterKleer PVC flexible tubing. The sensors first collected data from the tap water for approximately 12 minutes to establish a baseline before the water was intentionally contaminated with 1000 ppm lead, copper, iron, and zinc solutions made from ACS reagent grade hydrated metal salts, including lead (II) nitrate, iron (III) chloride hexahydrate, copper (II) nitrate trihydrate, and zinc nitrate hexahydrate. The metal solutions were added to the running tap water sequentially in doses of 1mL-8mL at a time using a glass pipette. These were chosen because they are soluble in water and dissociate, leading to metal ions in the aqueous solution. The salts undergo the following reactions in water

$$Pb(NO_3)_{2(s)} + H_2O_{(l)} \rightarrow Pb^{2+}_{(aq)} + 2NO^{-}_{3(aq)}$$

$$FeCl_{3(s)} + H_2O_{(l)} \rightarrow Fe^{3+}_{(aq)} + 3Cl^{-}_{(aq)}$$

$$Cu(NO_3)_{2(s)} + H_2O_{(l)} \rightarrow Cu^{2+}_{(aq)} + 2NO^{-}_{3(aq)}$$

$$Zn(NO_3)_{2(s)} + H_2O_{(l)} \rightarrow Zn^{2+}_{(aq)} + 2NO^{-}_{3(aq)}$$

Using a simple dilution calculation, the starting concentration of each contaminant when mixed into the flowing water would be 0.5,1,1.5,2,2.5,3,3.5,4 ppm. With dilution occurring as tap water flows into the system. A 45 second pause was given between each contaminant dose to allow for values of the system to return to steady state values.

Results and Discussion

The experimental procedures and data analysis methods described previously serve to gather data on the sensor accuracy/performance, asses the predictive abilities of the system to measure water quality, and determine the sensors' ability to detect contamination events for constituents of concern for point of use applications. The results for each phase of testing the wireless sensor network prototype are presented.

Sensor Verification Results

For the tap water samples which were taken from August 2021 through October 2021, sensor data from the pH, EC, temperature, and ORP sensors were collected and compared to the Myron Ultrameter multiparameter instrument. The comparison results between instruments were plotted for each parameter in Figures 4-7.

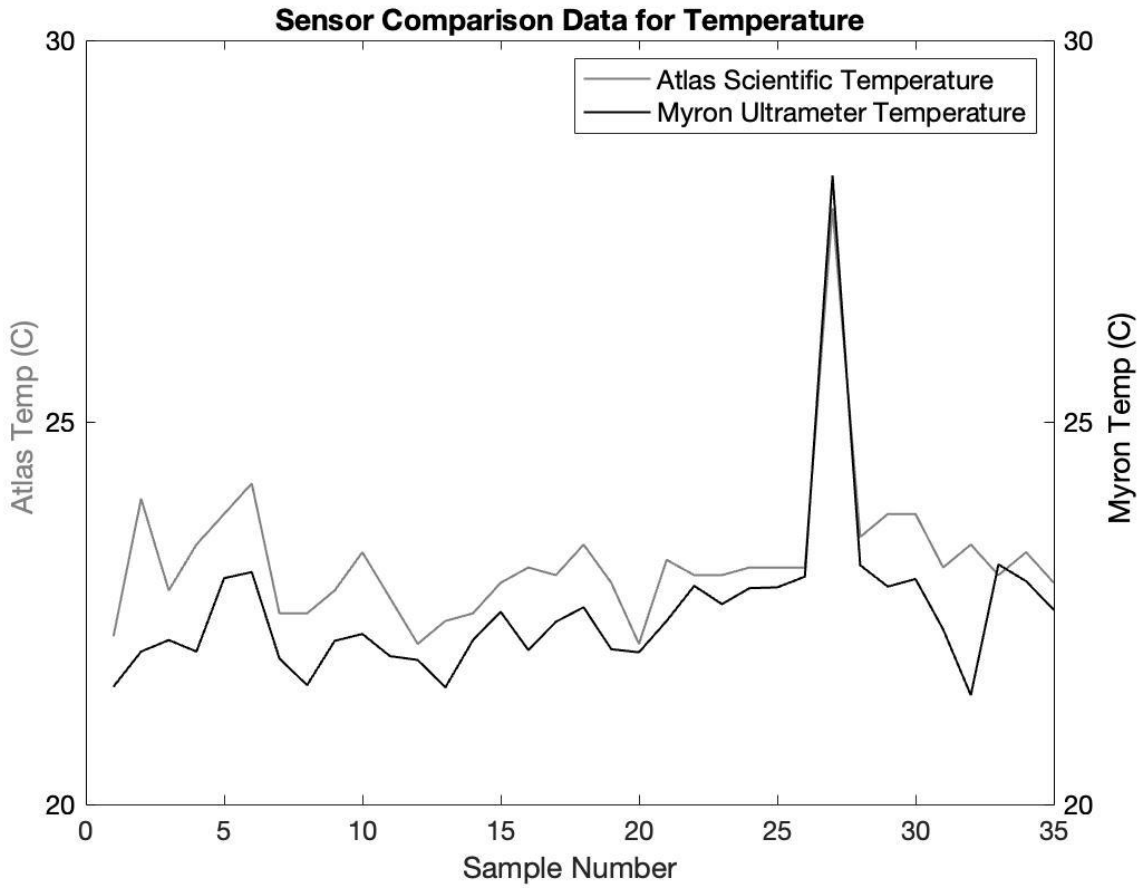

Figure 4: Temperature Instrument Comparison

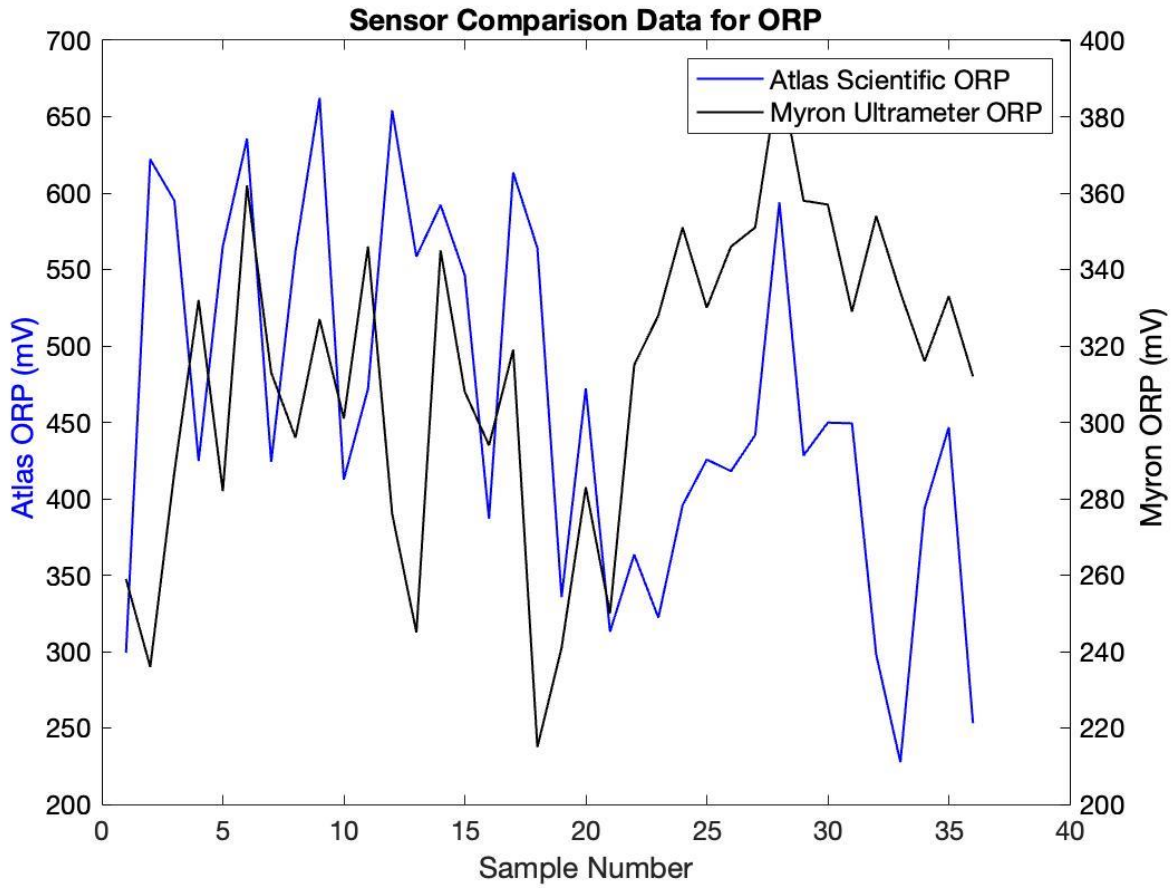

Figure 5: ORP Instrument Comparison

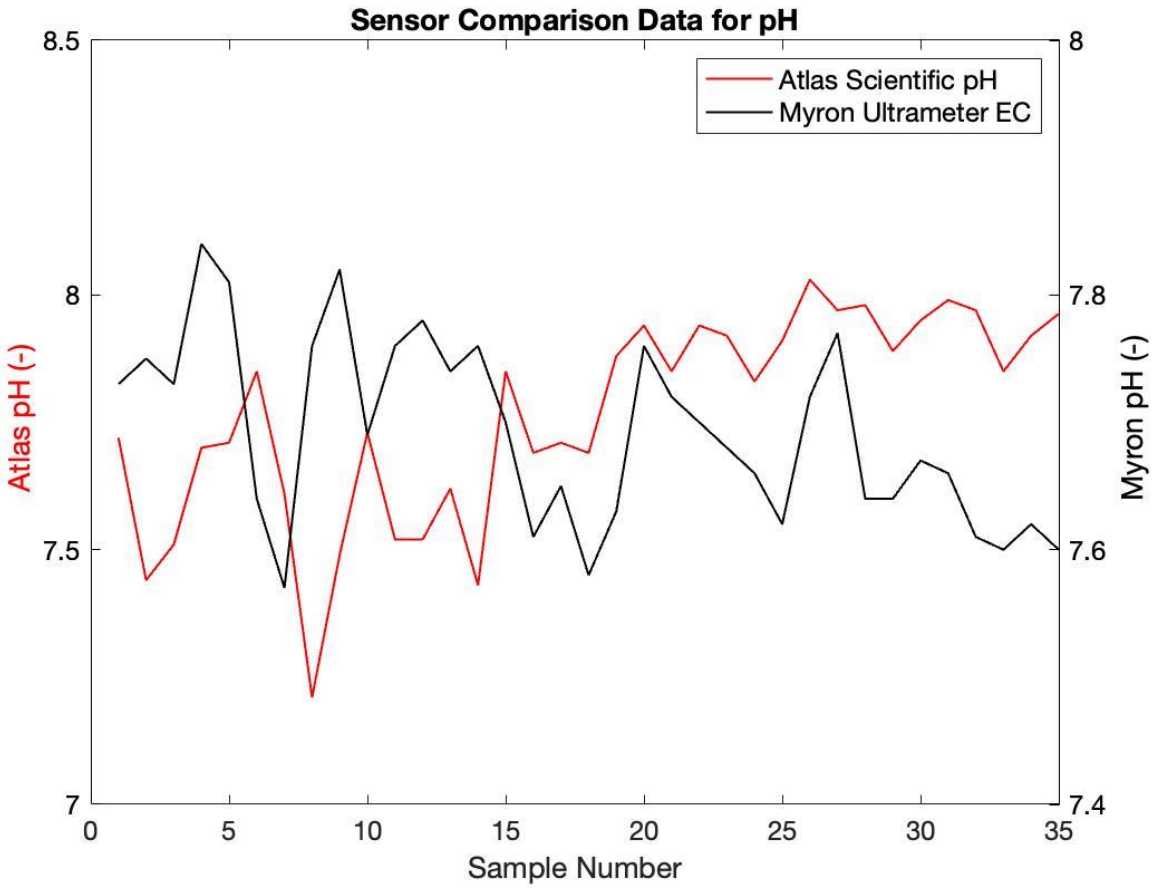

Figure 6: pH Instrument Comparison

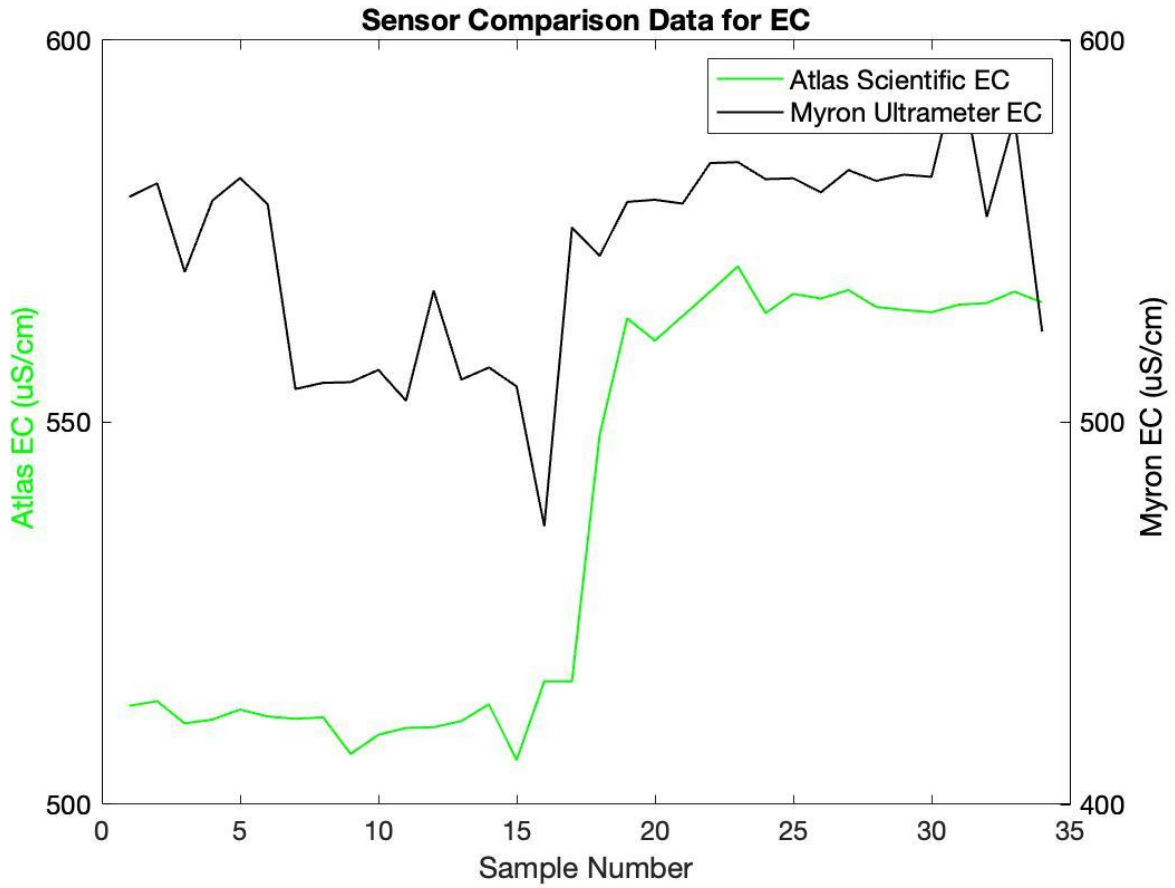

Figure 7: EC Instrument Comparison

Table 3:*Sensor Statistical F-Tests for Equal or Unequal Variance*

	Atlas Scientific pH	*Myron Ultrameter pH*	*Myron Ultrameter Temp*	*Atlas Scientific Temp*	*Myron UltraMeter EC*	*Atlas Scientific EC*	*Atlas Scientific ORP*	*Myron UltraMeter ORP*
Mean	7.8	7.7	22.5	23.2	537.9	545.1	461.6	311.8
Variance	0.040	0.005	1.231	0.903	742.779	736.899	14287.019	1699.621
Observations	35	35	35	35	34	34	36	36
df	34	34	34	34	33	33	35	35
F	7.656	-	1.364	-	1.008	-	8.406	-
P(F<=f) one-tail	2.148E-08	-	1.848E-01	-	4.910E-01	-	3.714E-09	-
F Critical one-tail	1.772E+00	-	1.772E+00	-	1.788E+00	-	1.757E+00	-

Table 4: *Sensor Statistical p-Tests for comparing instruments*

	Atlas Scientific pH	*Myron Ultrameter pH*	*Myron Ultrameter Temp*	*Atlas Scientific Temp*	*Myron UltraMeter pH*	*Atlas Scientific pH*	*Atlas Scientific ORP*	*Myron UltraMeter ORP*
Mean	7.77	7.69	22.53	23.19	537.91	545.06	461.60	311.75
Variance	0.04039	0.00528	1.23141	0.90257	742.77865	736.89898	14287.01942	1699.62143
Observations	35	35	35	35	34	34	36	36
Hypothesized Mean Difference	-	-	1.06699	-	-	-	-	-
Hypothesized Mean Difference	0	-	0	-	0	-	0	-
df	43	-	68	-	66	-	43	-
t Stat	2.03	-	-2.67	-	-1.08	-	7.11	-
P(T<=t) one-tail	2.44E-02	-	4.68E-03	-	1.41E-01	-	4.44E-09	-
t Critical one-tail	1.68	-	1.67	-	1.67	-	1.68	-
P(T<=t) two-tail	4.89E-02	-	9.37E-03	-	0.282	-	8.89E-09	-
t Critical two-tail	2.0167	-	1.9955	-	1.9966	-	2.0167	-

The results from the F-Test showed unequal variances for all sensors except temperature because $F > F_{Critical,one-tail}$ indicating that the p-test assuming unequal variances must be performed for pH, EC, and ORP sensors. The null hypothesis for comparing instruments states there is no significant difference between the performance of the sensors. To accept the null hypothesis the p-value for the two-tail p test must be less than or equal to $\alpha = 0.05$. Therefore, all the Atlas Scientific sensors, excluding EC (P(T<=t) two-tail > 0.05), were found to perform significantly differently from its Myron Ultrameter counterpart.

This result is expected for ORP sensors. YSI reports that the most common problem with ORP measurements for environmental water samples is that readings from various instruments for the same water sample can differ by a significant margin (50-100mV) even with the same sensor type and electronics, yet the sensors show identical or similar readings in ORP standards. This is explained by the fact that the tap water sampled is expected to be clean by most

standards. Therefore, it would likely contain few redox active species present and those that are present have low concentrations, which was confirmed by the results from the Free Chlorine testing of the samples as well. However, in standard solutions the two probes read the expected values, this is due to the concentration of redox-active species (such as ferricyanide/ferrocyanide in Zobells solution) being much higher. Therefore, it is suggested that historic data be used to help determine the validity of probes for environmental water samples and inconsistent data between probes does not always indicate a malfunctioning probe. As for the discrepancies between the pH values, it is believed that the issue may be due to sensor drift. This is presumed to be the issue due to the similar behavior patterns of the probe with the Atlas pH probe values becoming slightly shifted above the Myron Ultrameter values through the month of October 2021. This indicates more frequent calibration than recommended by the manufacturers may be necessary for long-term sensor deployment. To resolve the discrepancy between the pH probe performance it would be suggested to collect another set of tap water samples in the future with a recalibration frequency of every 2-4 weeks and determine if the new data allows for acceptance of the null hypothesis. Despite the difference in performance between the Atlas Scientific sensor probes and the Myron Ultrameter, the average values for the measured parameters are still in expected ranges for tap water.

Predictive Modeling Results

Data from the 5 sensors were also processed using Multivariate Regression Techniques to evaluate the ability to develop predictive models for the parameters listed in Table 5. The equations obtained for each parameter were calculated as:

$$DOC_{predicted} = 1.855 - 0.073\text{DO} + 8.52 * 10^{6}H^{+} + 0.00062\text{EC} + 0.039\text{Temperature} + 0.00046\text{ORP}$$

$$NTU_{predicted} = 0.104 - 0.022DO + 6.35 * 10^{6}H^{+} + 0.00015EC + 0.011Temperature + 0.00019ORP$$

$$Free\ Chlorine_{predicted} = -0.216 + 0.001DO - 1.86 * 10^{6}H^{+} + 0.00013EC + 0.007Temperature + 1.39 * 10^{-5}ORP$$

$$Copper_{predicted} = -0.007 - 0.002DO - 2.34 * 10^{6}H^{+} + 9.74 * 10^{-5}EC - 0.003Temperature + 0.00013ORP$$

$$Lead_{predicted} = 2.12 * 10^{-5} - 4.79 * 10^{-8}DO - 1.67 * 10^{2}H^{+} + 3.05 * 10^{8}EC - 7.72 * 10^{-7}Temperature - 1.47 * 10^{-8}ORP$$

$$Iron_{predicted} = 0.0004 - 1.203 * 10^{-5}DO - 5.01 * 10^{2}H^{+} - 4.52 * 10^{-7}EC - 1.79 * 10^{-7}Temperature - 5.62 * 10^{-8}ORP$$

$$Zinc_{predicted} = 0.066 - 0.0006DO - 1.32 * 10^{5}H^{+} - 7.05 * 10^{-5}EC - 0.001Temperature + 1.54 * 10^{-5}ORP$$

$$Manganese_{predicted} = 3.89 * 10^{-5} - 1.54 * 10^{-6}DO + 138.19H^{+} - 3.30 * 10^{-8}EC + 1.70 * 10^{-7}Temperature - 1.59 * 10^{-8}ORP$$

The equations were cross checked using both the Excel data analysis Toolpak and the "mldivide" function in MATLAB to verify the solutions for each system of linear equations. The predicted and measured values are plotted in Figures 8-15

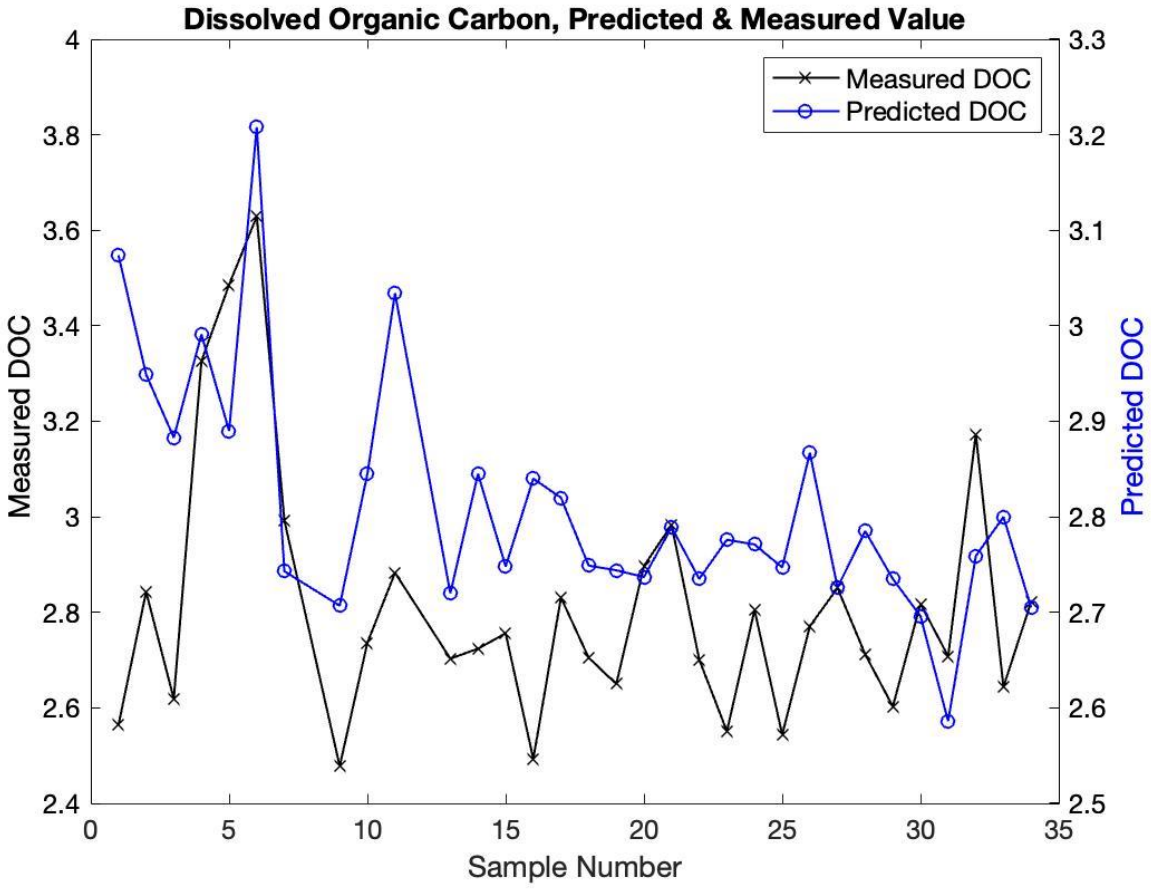

Figure 8: Predicted vs. Measured DOC

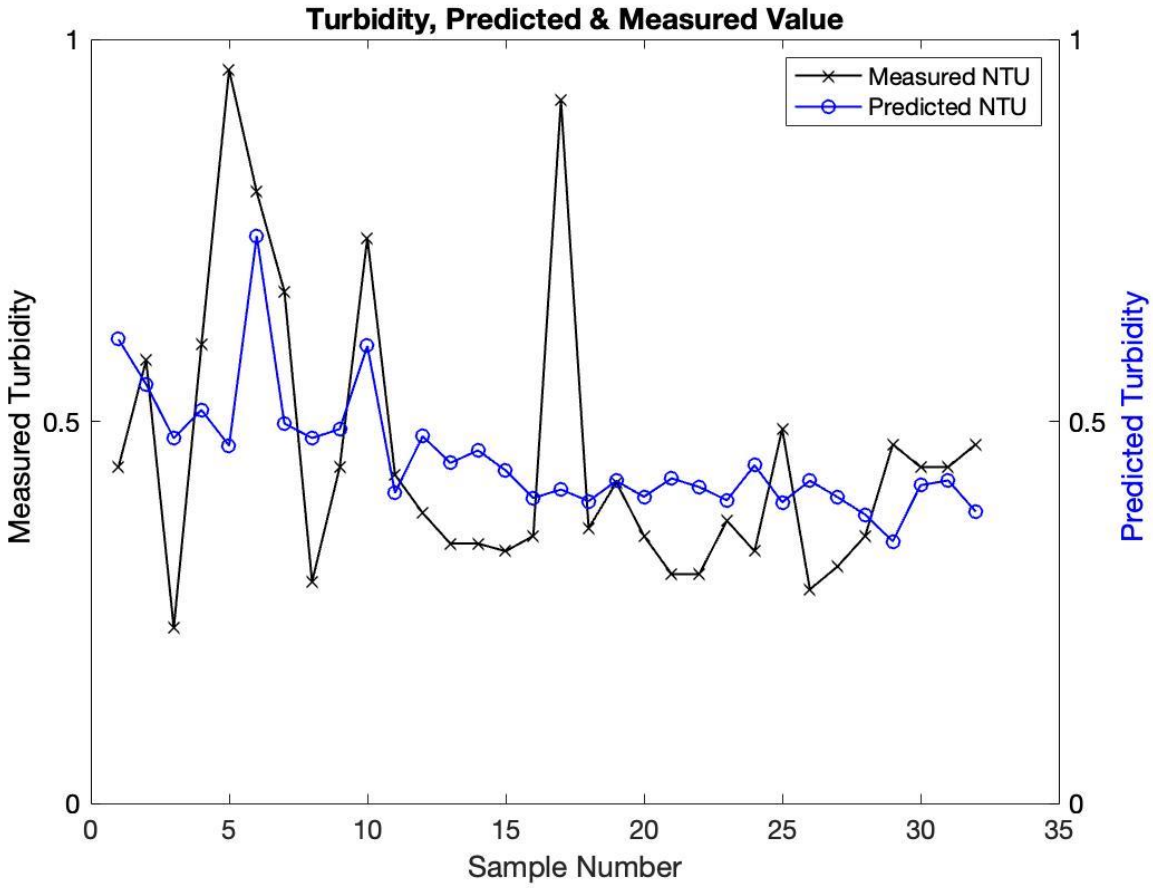

Figure 9: Predicted vs. Measured Turbidity

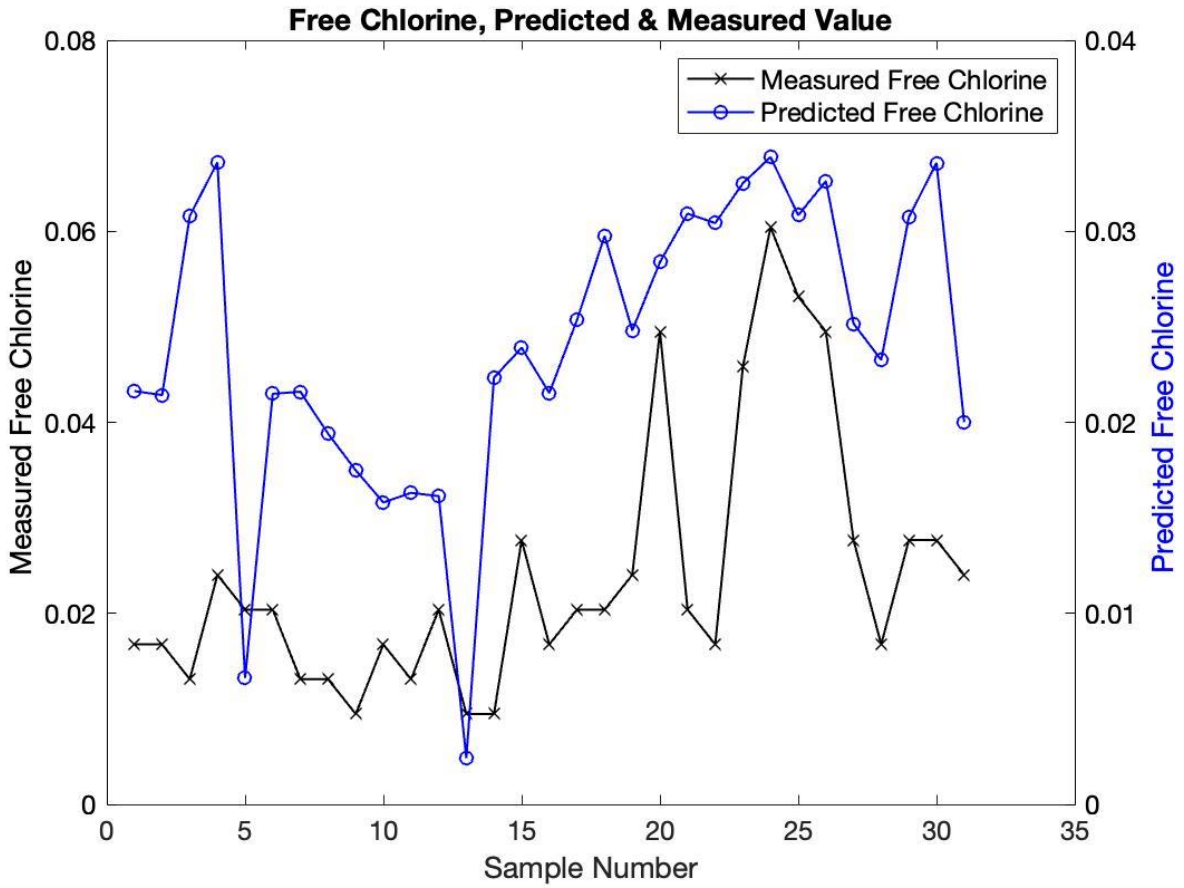

Figure 10: Predicted vs. Measured Free Chlorine

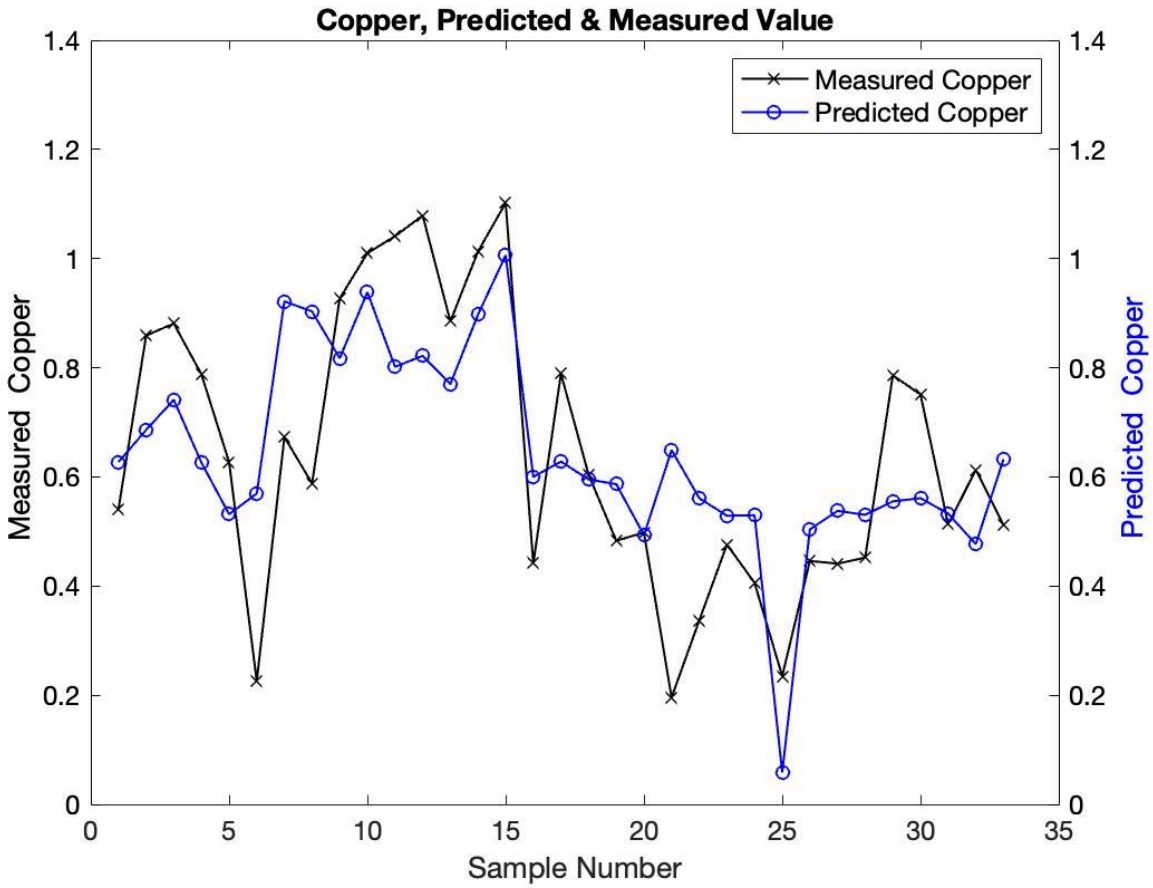

Figure 11: Predicted vs. Measured Copper

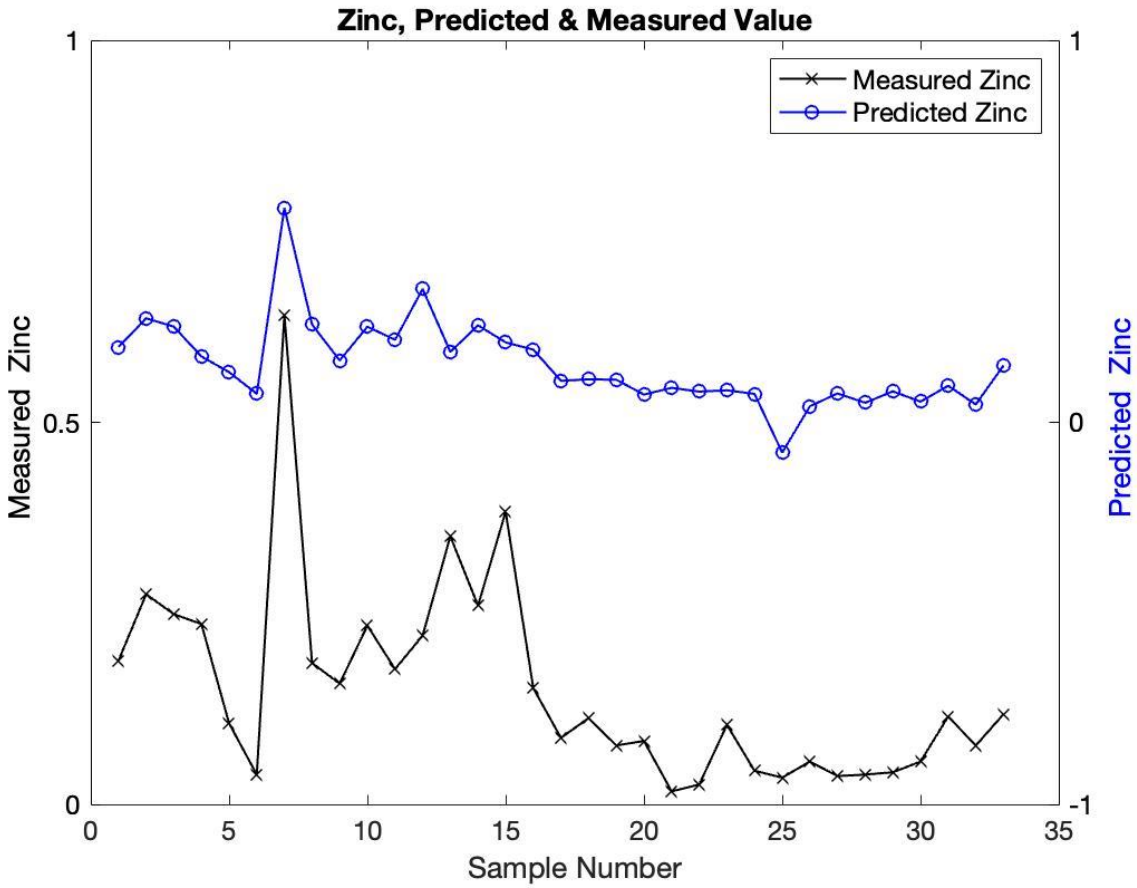

Figure 12: Predicted vs. Measured Zinc

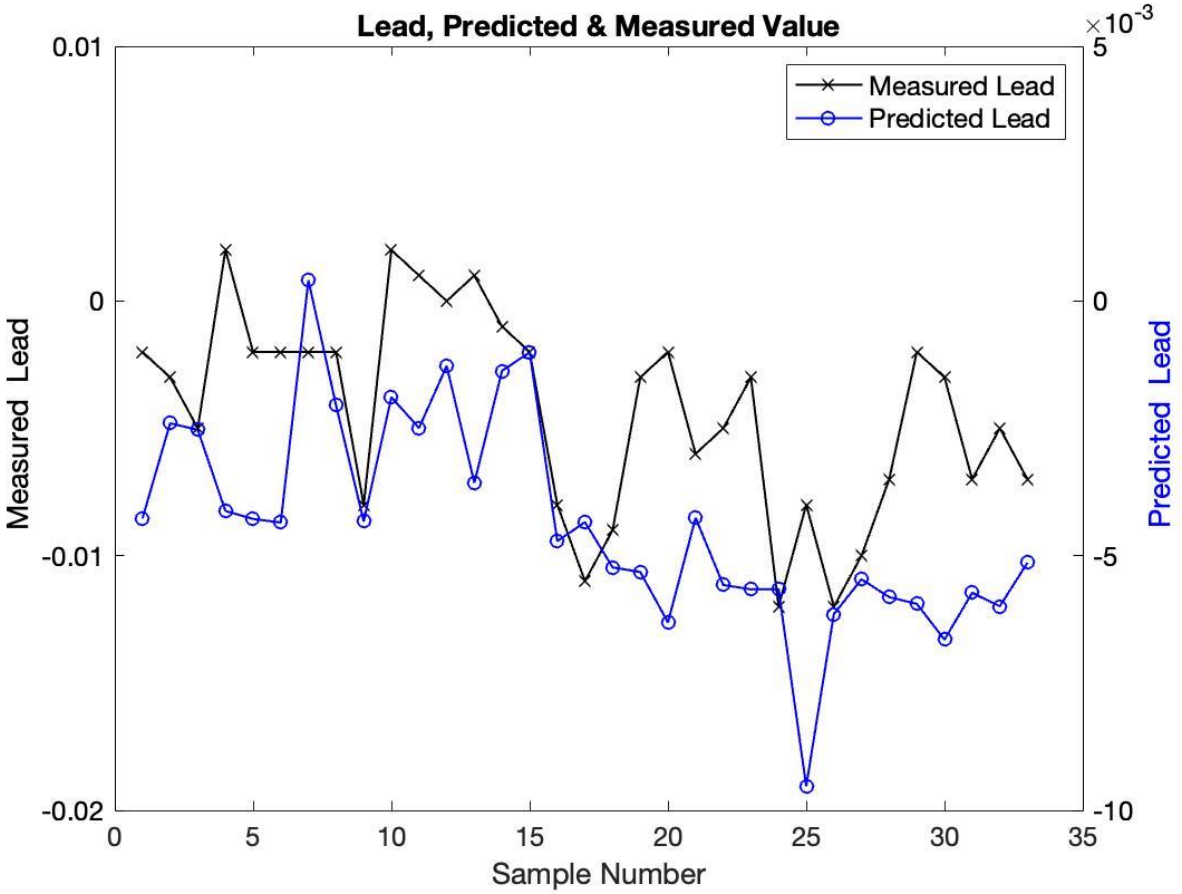

Figure 13: Predicted vs. Measured Lead

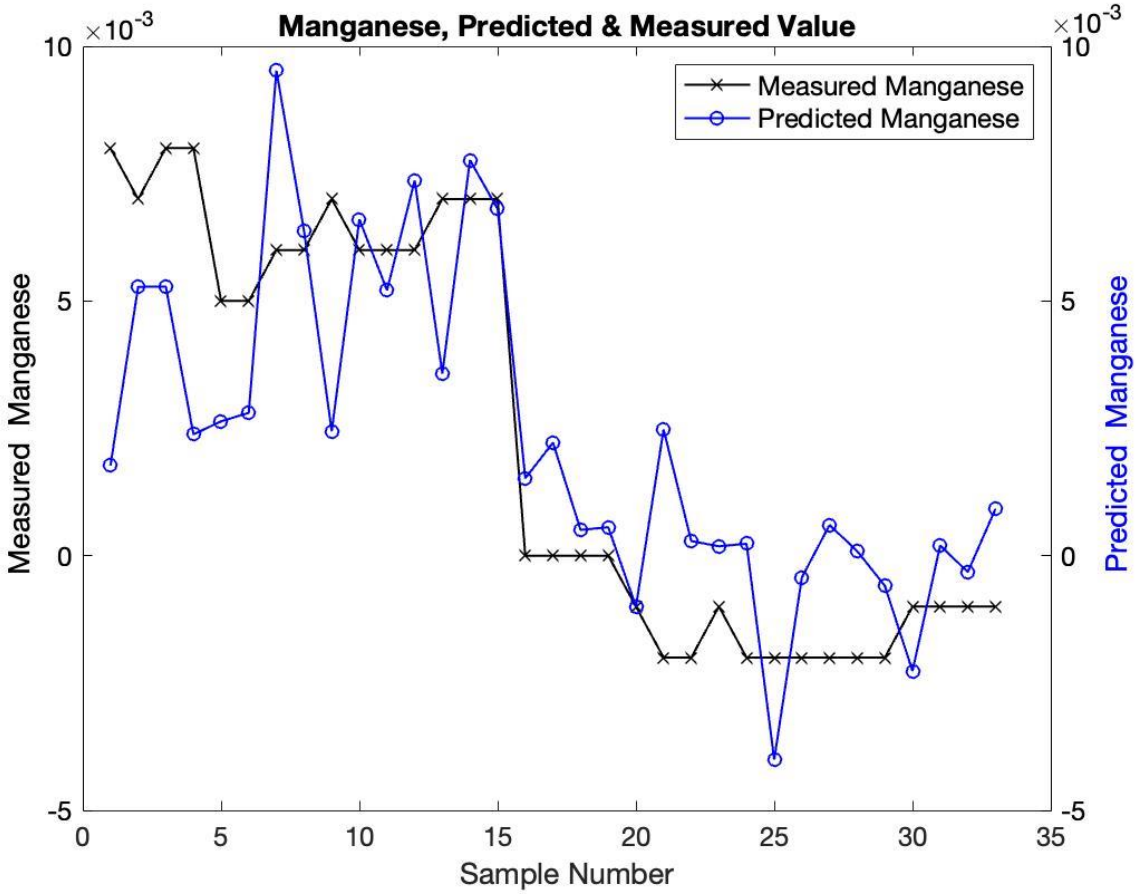

Figure 14: Predicted vs. Measured Manganese

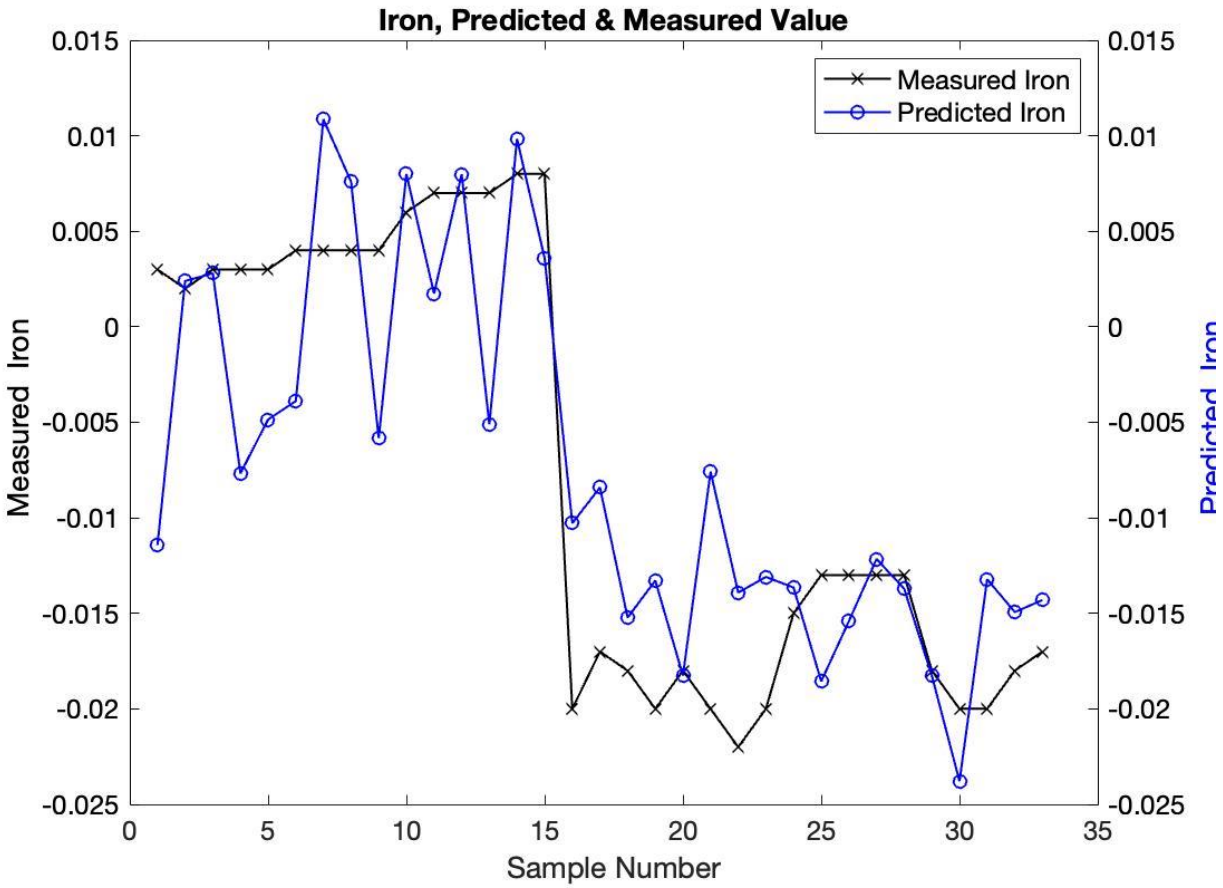

Figure 15: Predicted vs. Measured Iron

The following regression statistics were obtained for each predicted variable:

Table 5: *Regression Statistics for Sensor Modelling*

	DOC	*Turbidity*	*Free Chlorine*	*Copper*	*Lead*	*Zinc*	*Manganese*	*Iron*
Multiple R	0.47	0.44	0.57	0.71	0.51	0.88	0.78	0.81037973
R Square	0.22	0.19	0.32	0.50	0.26	0.77	0.61	0.6567153
Adjusted R Square	0.07	0.03	0.19	0.41	0.13	0.73	0.54	0.59314406
Standard Error	0.26	0.18	0.01	0.20	3.69E-03	0.07	2.76E-03	0.00739343
RMSE	0.23	0.16	0.01	0.18	3.34E-03	0.06	2.50E-03	0.01180409
Observations	32	32	31	33	33	33	33	33

The zinc correlations had the best results for both the correlation coefficient and the adjusted coefficient of determination indicating a strong linear relationship for the model. The adjusted coefficient also indicates that 73% of the variation of the zinc values around the mean are explained by the sensor inputs. Of the metal ions, it is observed that the models for lead performed the worst, this is likely due to the low lead concentration in the water leading to negative values obtained for some samples from the ICP-OES, indicating that many of the samples had lead values below the detection limit, which may cause erroneous results. Multiple regression coefficients can also be reduced due to collinearity between the input variables. This was tested using the "correl" function in excel to create a correlation matrix demonstrated in Table 6. Some correlation was observed between pH and EC and DO and ORP, however this should not affect the predictive capabilities of the models, and the correlations are low enough ($R^2 < 0.6$) to still satisfy the assumption of independence between explanatory variables to meet the model requirements.

Table 6*: Correlation Matrix for Atlas Scientific Sensors*

	DO 97 (mg/L)	*pH 99 ()*	*EC 100 (uS)*	*RTD 102 (c)*	*ORP 98 (mV)*
DO 97 (mg/L)	1	-	-	-	-
pH 99 ()	-0.1004527	1	-	-	-
EC 100 (uS)	-0.3146643	0.58183886	1	-	-
RTD 102 (c)	0.27497674	0.34699723	0.17328755	1	-
ORP 98 (mV)	0.38292437	-0.6047262	-0.4330551	0.06550297	1

Contaminant Dosing Results

Lastly, the metal dosing was conducted. The last 8 minutes of each run represents the timeframe when the Metal Solutions were introduced. The "isoutlier" function in MATLAB was used to calculate upper and lower thresholds for each sensor. The program uses a True/False

logic array to determine if a value is outside either range and marks values that are determined as outliers with and “x.” The sensors which recognized outliers are plotted in Figures 16-26. The pH results with low values found for outlier detections can be attributed to the acidity of the metal solutions used for dosing which ranged in pH's from approximately 3.5-6. Based on the point in time at which outliers were detected it appears that the sensors detected the lowest concentrations for copper followed by zinc, then lead, and finally iron. EC appears to have some false alarms for the zinc and iron contaminations, as values are detected before the metal contamination was introduced into the flowing water. Otherwise, the EC sensors appear to have responded well to the introduction of zinc and sodium chloride as many outliers are detected after the additions of those contaminants. Lastly, many outliers were detected for DO upon the addition of sodium chloride however they would be expected to decrease because increasing the salinity reduces oxygen solubility indicating bad performance of the DO sensor for anomaly detection.

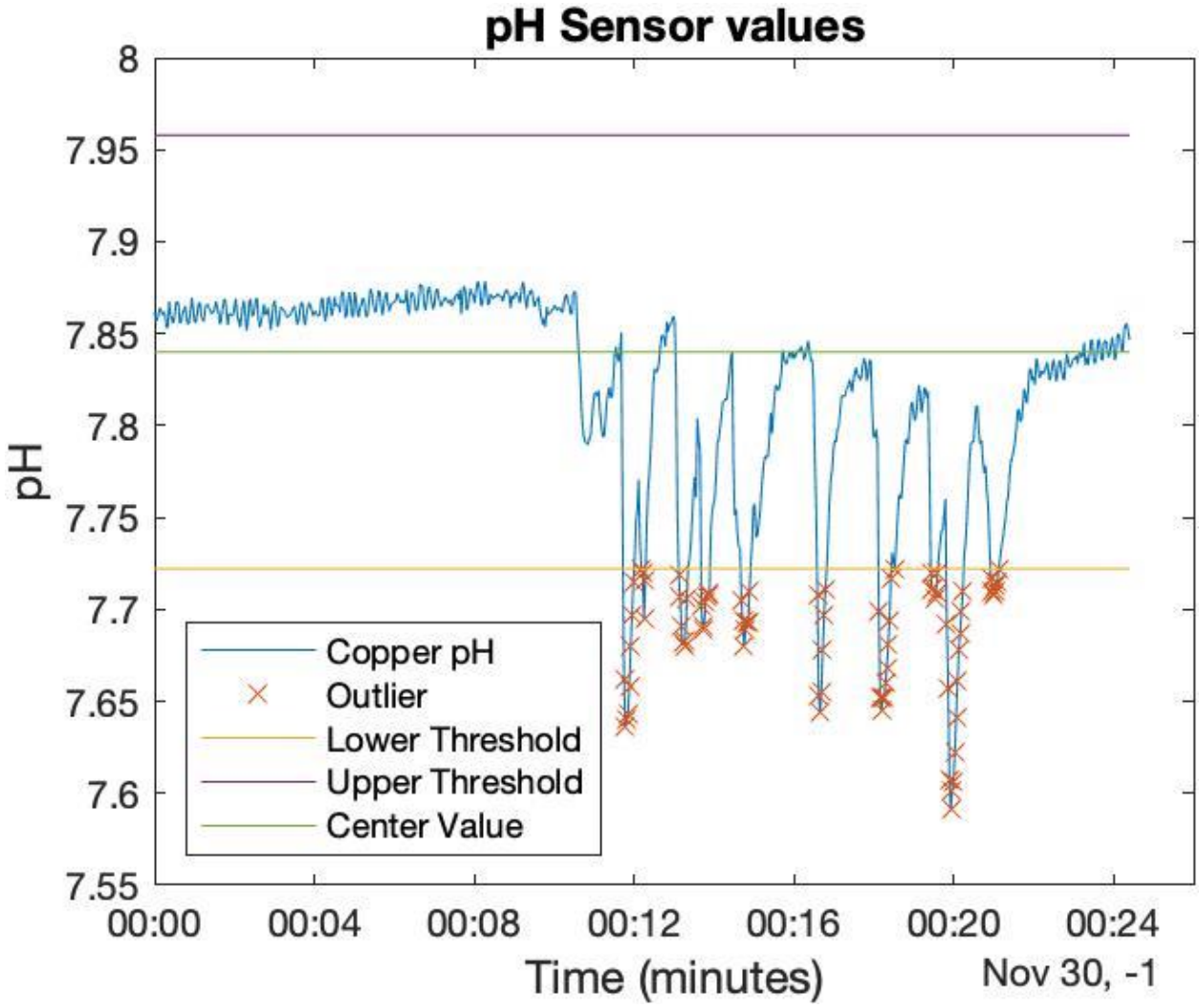

Figure 16: pH Response to Copper Contamination

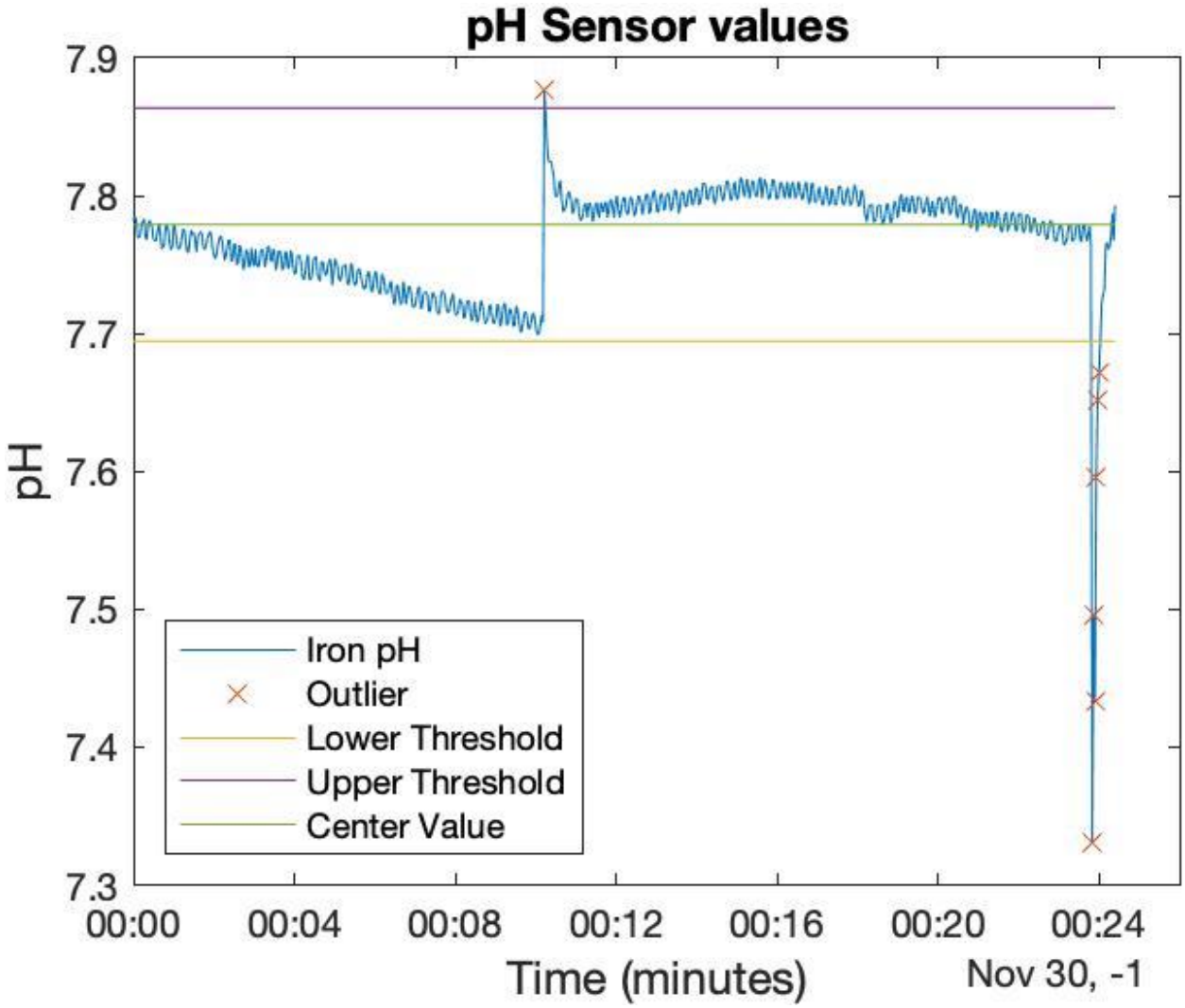

Figure 17: pH Response to Iron Contamination

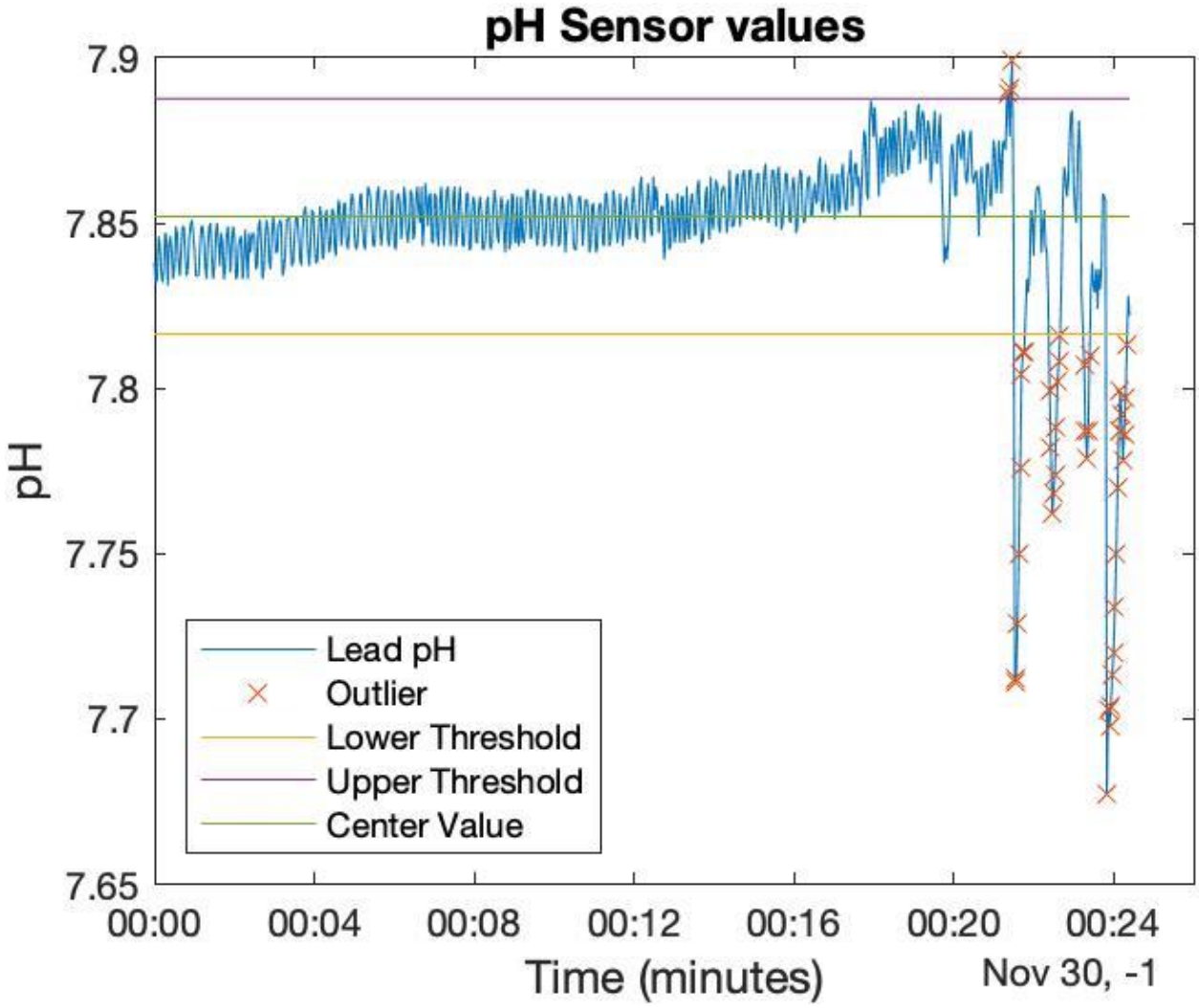

Figure 18: pH Response to Lead Contamination

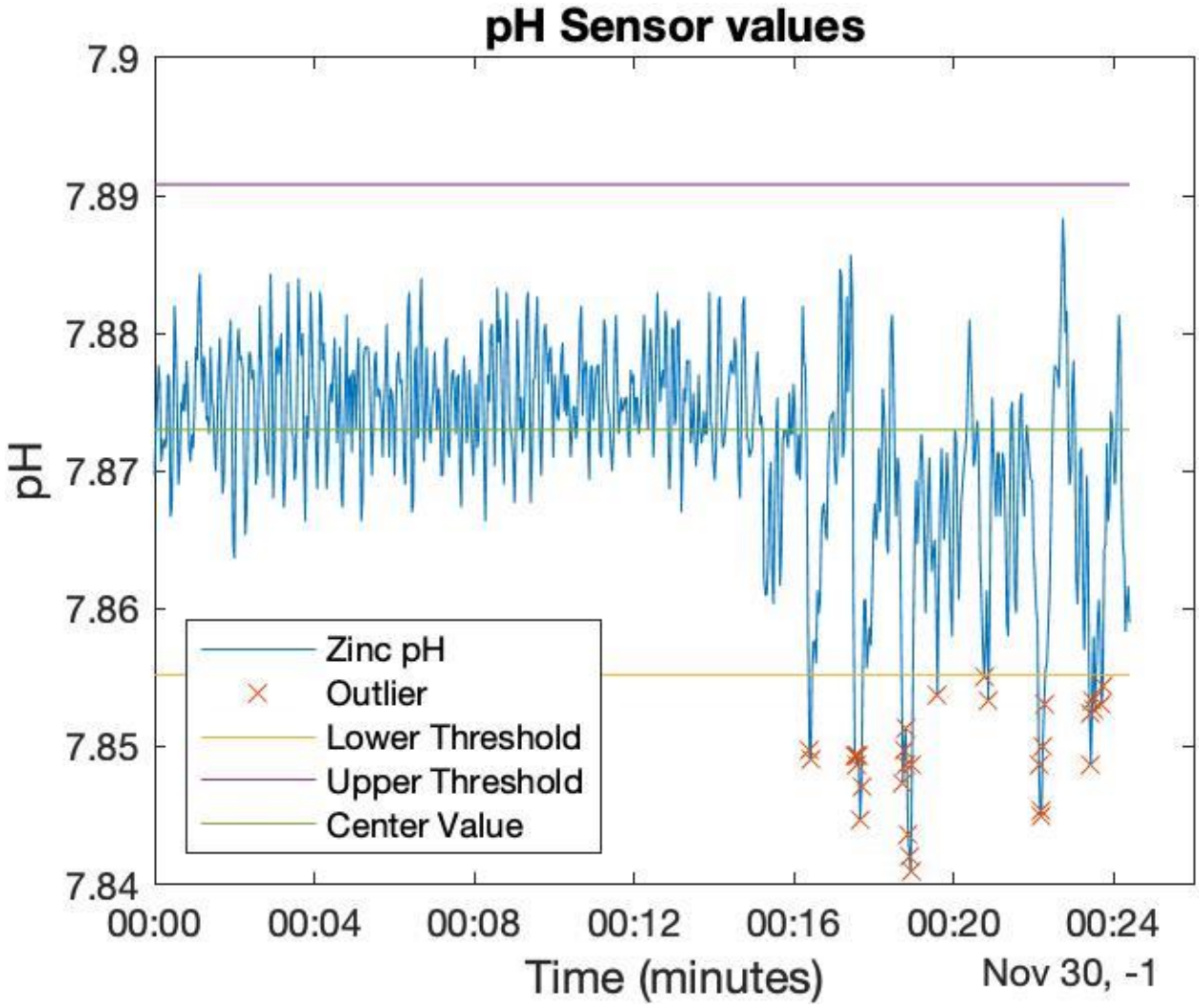

Figure 19: pH Response to Zinc Contamination

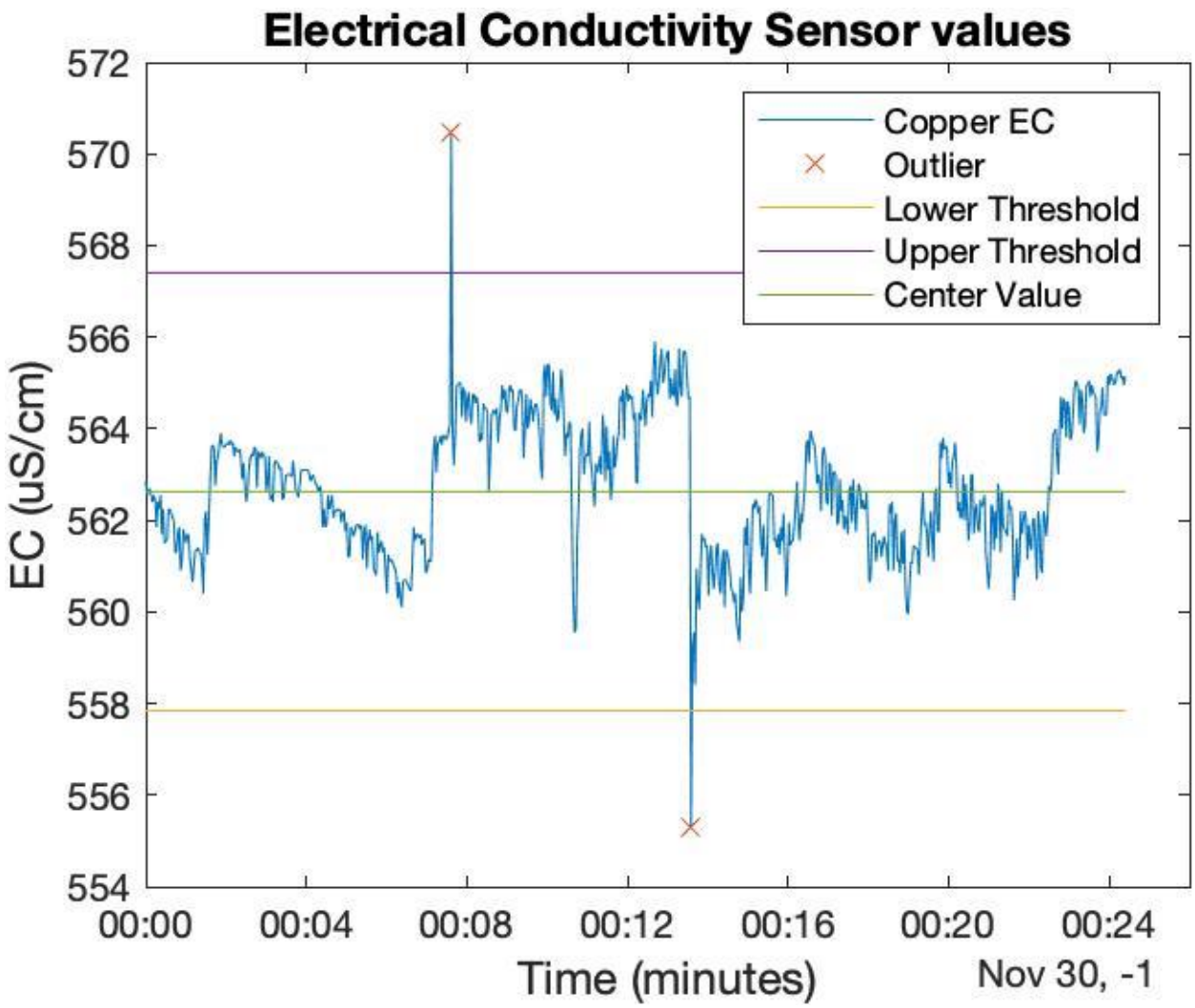

Figure 20: EC Response to Copper Contamination

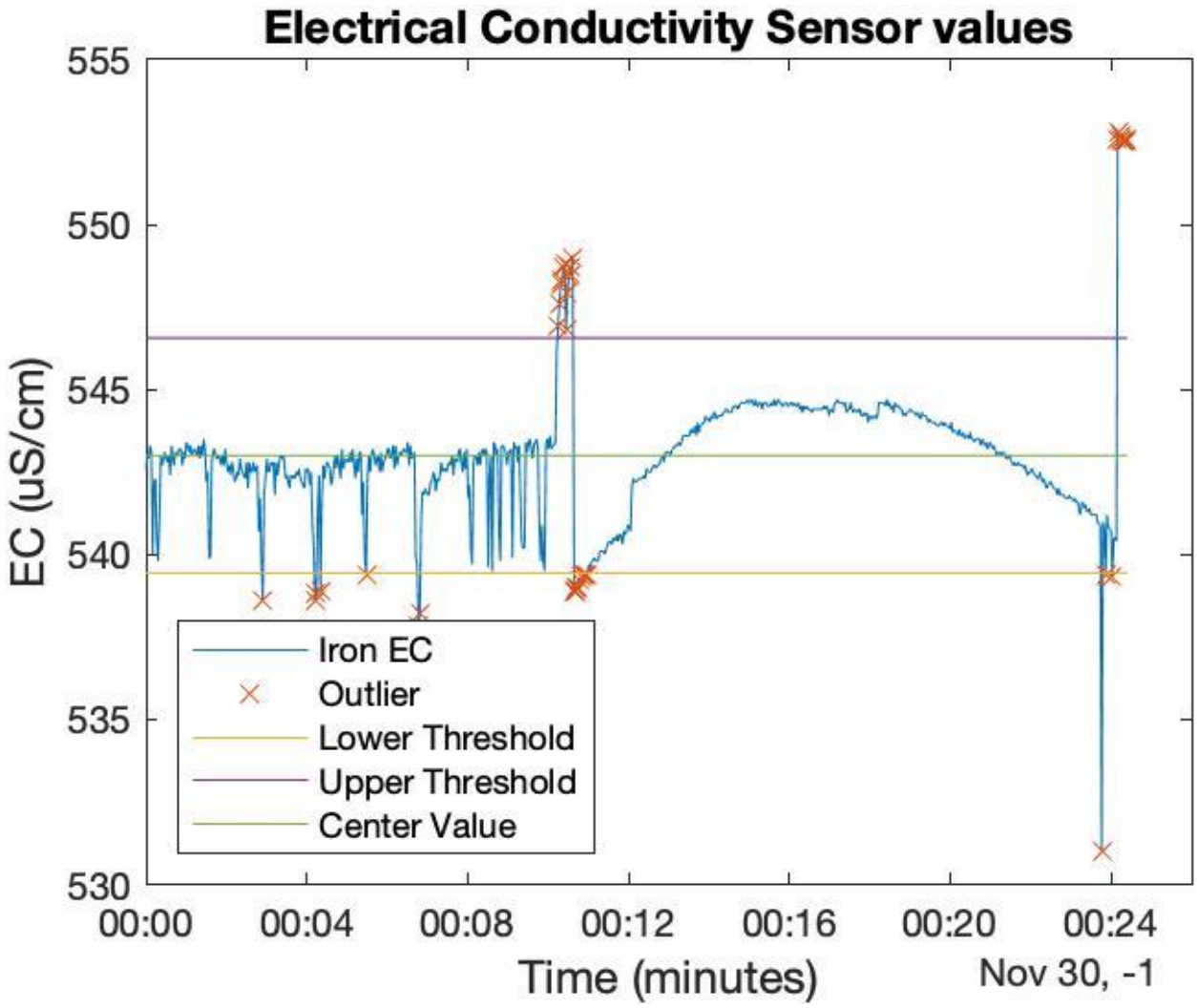

Figure 21: EC Response to Iron Contamination

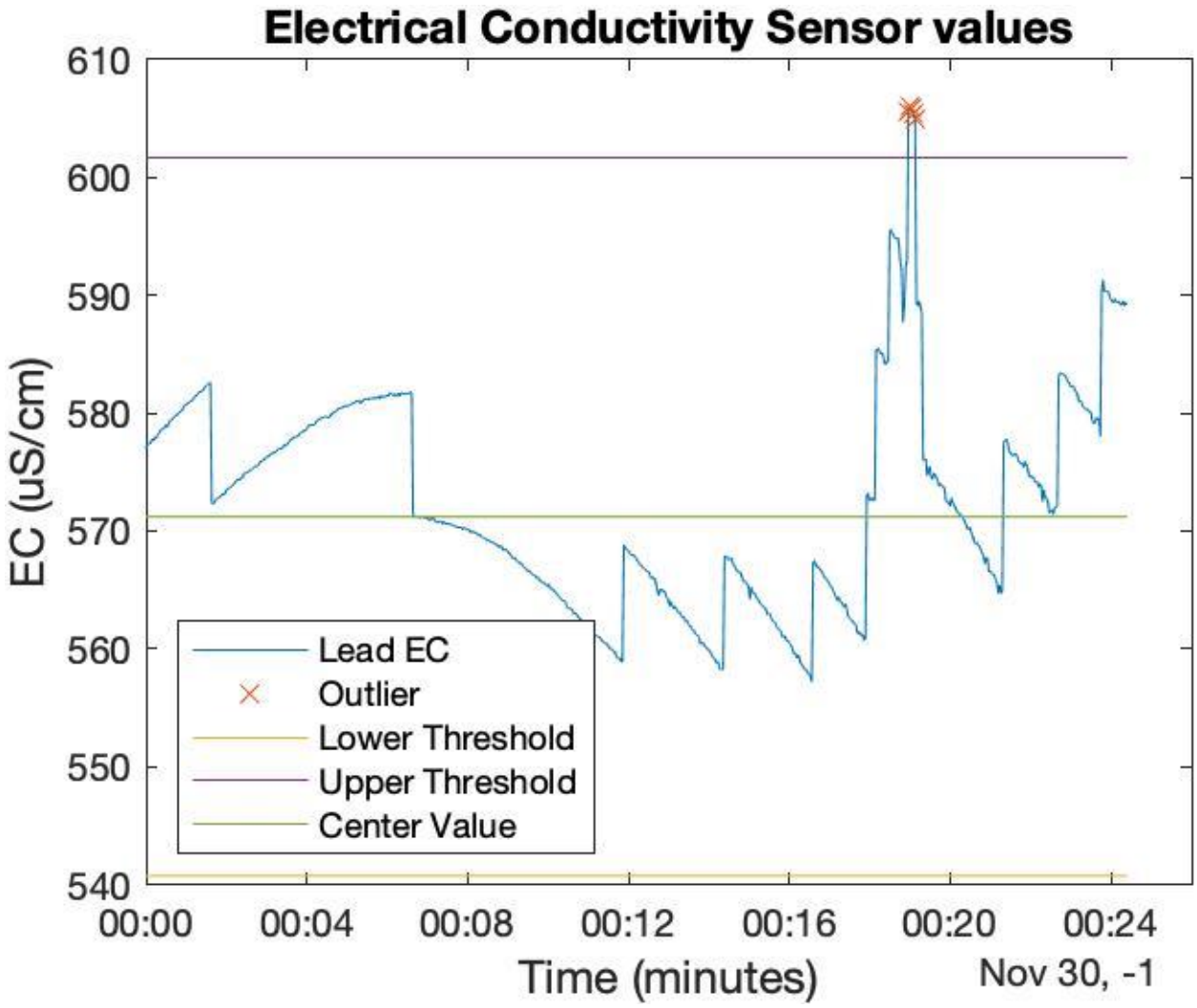

Figure 22: EC Response to Lead Contamination

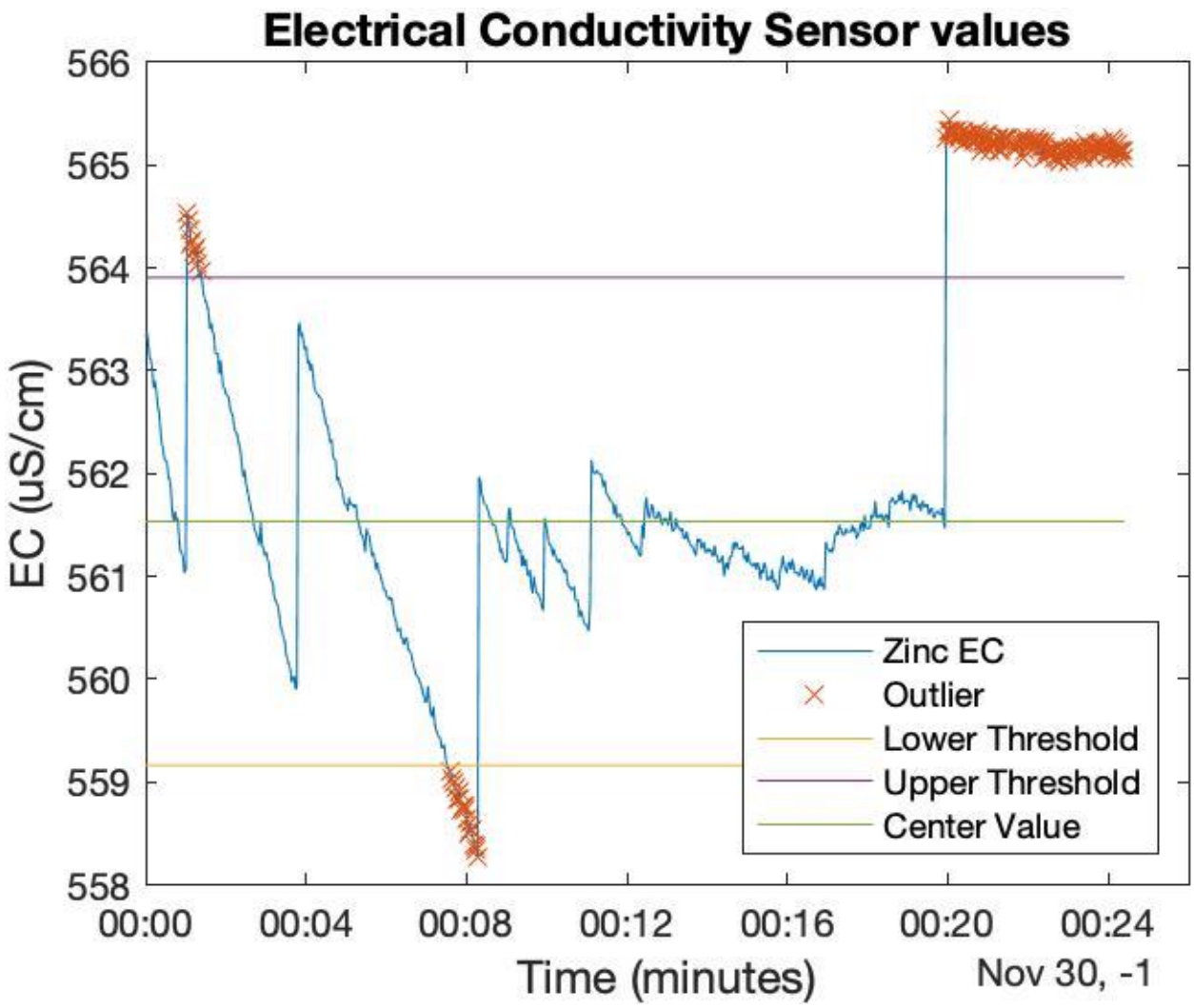

Figure 23: EC Response to Zinc Contamination

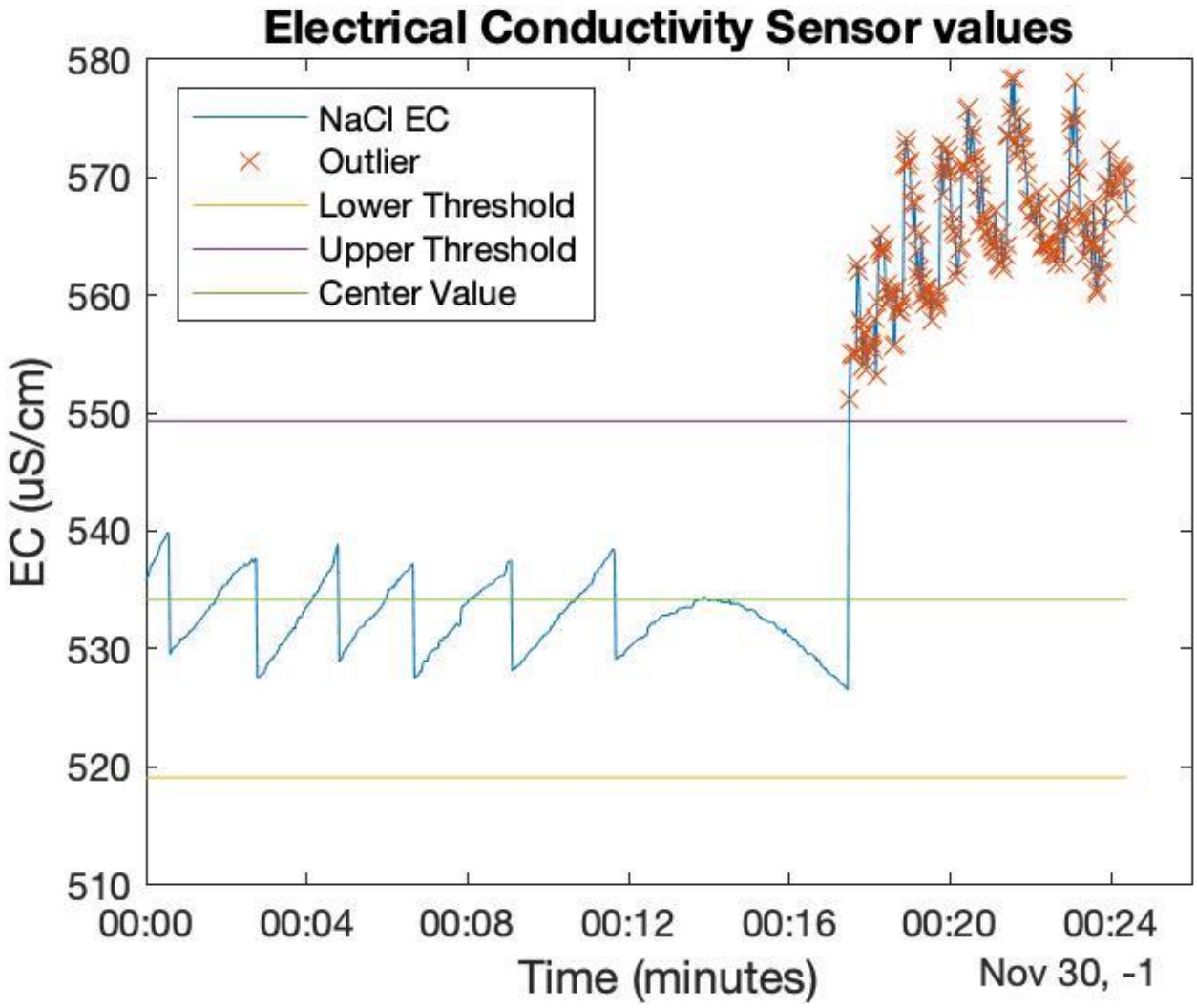

Figure 24: EC Response to NaCl Contamination

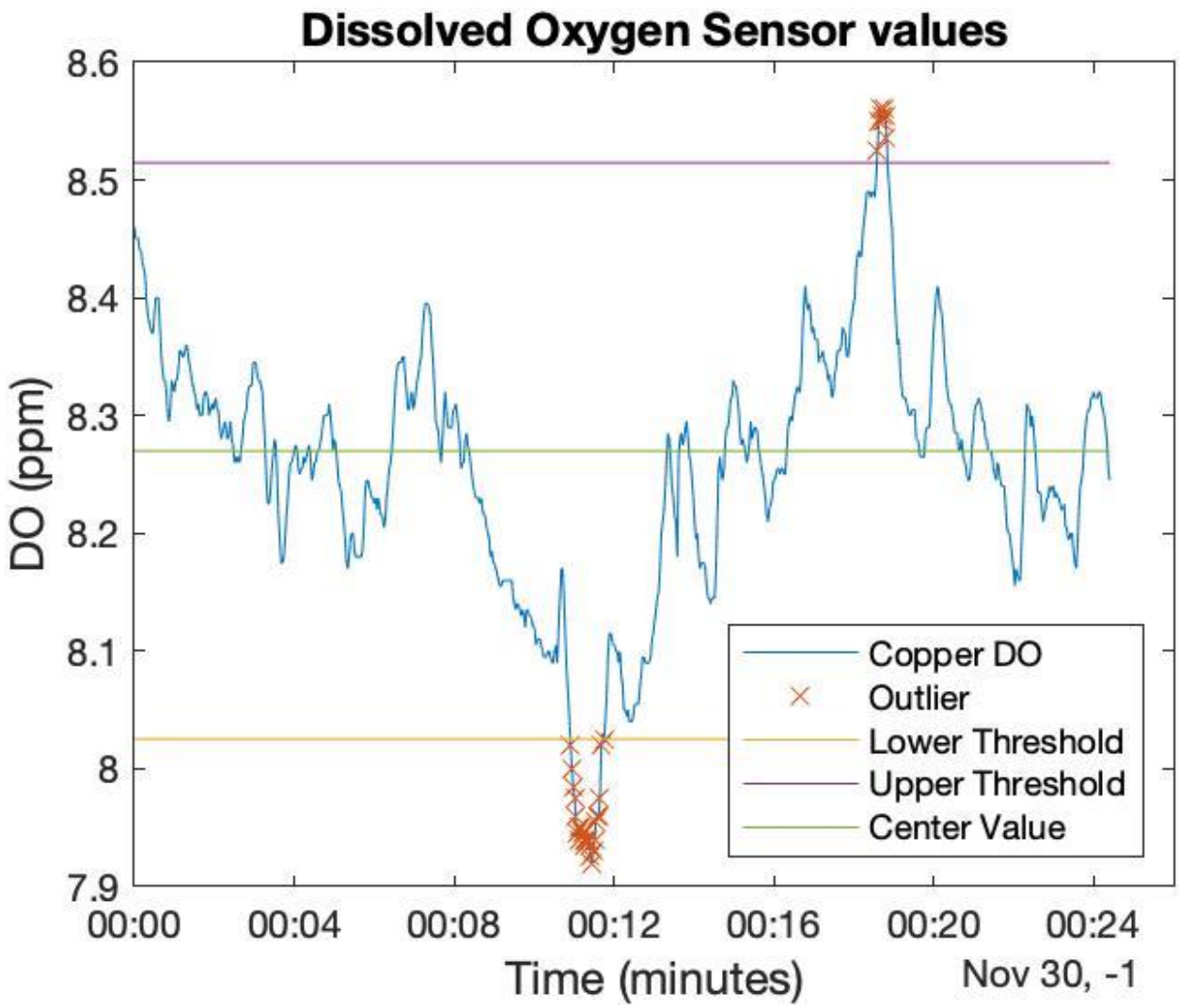

Figure 25: DO Response to Copper Contamination

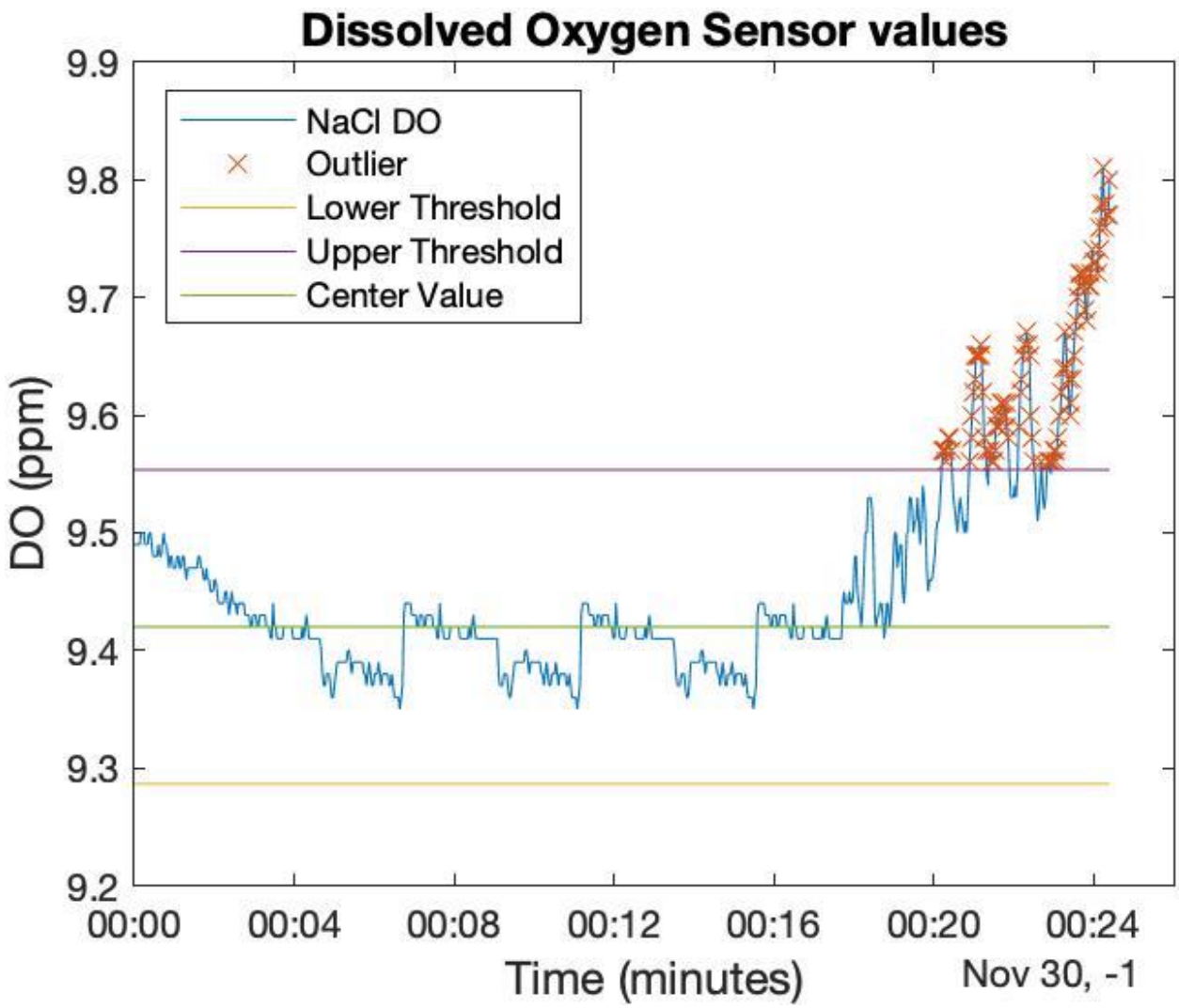

Figure 26: DO Response to NaCl Contamination

Conclusions and Future Work

The design and prototype proposed present a wireless water quality monitoring system for point of use applications using commercially available sensor probes. The system architecture is built on a Raspberry Pi microprocessor and includes software that allows for remote access of sensor data for users and operators. The experimental results determine the ability of the system to predict other primary and secondary drinking water standards with satisfactory RMSE values for the dataset that was tested, as well as recognize outliers caused by contamination events. The project could be expanded upon by integrating the predictive models

into automated data processing scripts and generally making the system more autonomous. The proposed algorithms can be improved upon with more robust data analysis and machine learning techniques due to the complexity of a data set with five explanatory variables. Future work includes retrofitting the sensor prototype into mobile water filtration units deployed in areas with known water quality issues and developing an app to accompany the system which alerts users of water quality issue when they are detected. Overall, the viability of the system is promising for combining many of the advantages of wireless sensor networks for remote monitoring of drinking water.

Appendix A

Data Collection Python Script

```
# port over to python3

import os
import sys
import time
import signal
from collections import OrderedDict
import pandas as pd
from subprocess import Popen, PIPE, STDOUT
from enum import Enum
import matplotlib.pyplot as plt

class Log(Enum):
    DEBUG = 1
    INFO = 2

# Note logMode does not affect data collection
logMode = Log.DEBUG  # DEBUG mode prints the most to the screen, INFO prints
only important information
device_types = ['DO 97', 'ORP 98', 'pH 99', 'EC 100', 'RTD 102']
sensor_reading_key = "Success"
temp_tolerance_before_update = 1  # How much the temperature has to change
before updating sensors
set_temp = 25
device_units = {'DO 97': 'mg/L', 'ORP 98': 'mV', 'pH 99': '', 'EC 100': 'uS',
'RTD 102': 'c'}

IS_PLOT = True
COMMENTS = ""

def write_data_to_file(data, file):
    print("Writing data: " + data + " to file: " + file)
    f = open(file, "a")  # Opens file in append mode (i.e. won't overwrite
prev info)
    f.write(data)
    f.close()

def create_file_if_no_existing(file):
    if os.path.isfile(file):
        if logMode == Log.DEBUG:
            print("Data file: " + file + " exists... retaining old file")
    else:
        f = open(file, "w")  # Since file doesn't exist, creating new one
        if logMode == Log.DEBUG:
            print("Data file: " + file + " does not exist... creating new
file")
        f.close()

def check_data_validity(data):
    try:
        data = float(data)
        return True
```

```
    except:
        return False

def process_atlas_response(response, p, data_log):
    # Check if response from Atlas software is a sensor reading (all sensor
readings contain this key in it)
    if response.find(sensor_reading_key) != -1:
        if logMode == Log.DEBUG:
            print("Data found")

        for device in device_types:
            if response.find(device) != -1:  # Check if specific device is
the one we are writing to
                device_key = device + ' (' + device_units[device] + ')'
                if device_key not in data_log:
                    data_log[device_key] = []

                if response.count(':') != 1: # Ignore the input if there's
two ':' as outputs can get combined with resetting values
                    data_log[device_key].append(None)
                    return

                # Parse through sample code output
                data_location = response.find(":")  # The response format has
data following a ':'
                data = response[data_location + 2:]
                if not check_data_validity(data):
                    return

                # Add device reading
                data_log[device_key].append(data.rstrip())

                # Plot in real time
                if IS_PLOT:
                    plt.figure(device)
                    plt.plot(range(len(data_log[device_key])),
data_log[device_key])
                    plt.draw()
                    plt.pause(0.01)

                if device == "RTD 102":
                    # If a temperature is passed in, we want to update the
temperature of all devices
                    check_temperature(data, p)
    elif response.find('-------press ctrl-c to stop the polling') != -1:
        data_log['Date'].append(time.strftime('%m/%d/%Y'))
        data_log['Time'].append(time.strftime("%H:%M:%S"))

def prep_devices(p):
    for device in device_types:
        if (device == "RTD 102"):
            device_num = device[device.find(' '):]
            units = device_units[device]
            units_command = device_num + ":S," + units + "\n"
            if logMode == Log.DEBUG:
                print(units_command)
            p.stdin.write(units_command)
```

```
            if logMode == Log.DEBUG:
                print("Updated units of device: " + device + " to: " + units)

def start_up(p):
    if logMode == Log.DEBUG:
        global IS_PLOT
        IS_PLOT = raw_input("Require plotting of data? (y/n)")
        if IS_PLOT.lower() == 'y':
            IS_PLOT = True
        else:
            IS_PLOT = False
        global COMMENTS
        COMMENTS = raw_input("Add Comments: ")

    if logMode == Log.DEBUG:
        print("Initializing all temperatures to: " + str(set_temp))

    update_temperature(set_temp, p)
    prep_devices(p)


def update_temperature(temp, p):
    for device in device_types:
        if device != "RTD 102":  # Only want to update the temp setting of
non-temp sensors
            device_num = device[device.find(' '):]
            temp_update_command = device_num + ":T," + str(temp) + "\n"
            if logMode == Log.DEBUG:
                print(temp_update_command)
            p.stdin.write(temp_update_command)
            if logMode == Log.DEBUG:
                print("Updated temperature of device: " + device + " to: " +
str(temp))

def check_temperature(temp, p):
    global set_temp
    # If the temperature is outside of the acceptable range of differences,
we update the temp
    try:
        if abs(float(temp) - float(set_temp)) > temp_tolerance_before_update:
            # Stop collecting data
            p.send_signal(signal.SIGINT)
            set_temp = temp
            update_temperature(temp, p)
            p.stdin.write(b'Poll\n')
    except:
        print('Temp Value was not read')

def main():
    # Update the time and date of the system to be correct for logging
purposes
    os.system("sudo date -s \"$(curl -s --head http://google.com | grep
^Date: | sed 's/Date: //g')\"")

    # path of the atlas scientific code to run
    # TODO change this to be in a separate folder and add Atlas Code to Git
Repository
```

```
    script_path = './Raspberry-Pi-sample-code/i2c.py'
    # Runs the Atlas Code to get data from sensors
    p = Popen([sys.executable, '-u', script_path], stdin=PIPE,
              stdout=PIPE, stderr=STDOUT, bufsize=1)
    start_up(p)
    print("Starting Atlas Scientific Sensor Monitoring")
    p.stdin.write(b'Poll\n')

    data_log = OrderedDict()
    data_log['Date'] = []
    data_log['Time'] = []
    with p.stdout:
        try:
            for line in iter(p.stdout.readline, b''):
                if logMode == Log.DEBUG:
                    print("Received data: \n" + line)
                process_atlas_response(line, p, data_log)
        except KeyboardInterrupt:
            # Add comments from start up

            # Use pandas dataframe to convert to csv after exiting with
Ctrl+C
            series_list = []
            for column in data_log.keys():
                series_list.append(pd.Series(data_log[column], name =
column))

            df = pd.concat(series_list, axis=1)

            # Ask whether to keep data set and append comments to first row
            if raw_input('Keep data set? (y/n) ').lower() == 'y':
                df['Keep?'] = 'Yes'
            else:
                df['Keep?'] = 'No'
            df['Comments'] = ''
            df['Comments'][0] = COMMENTS

            df.to_csv('./data/' + time.strftime("%m-%d-%Y %H:%M:%S") +
".csv", index=False, header=True)
            print('Finished writing to ./data/' + time.strftime("%m-%d-%Y
%H:%M:%S") + '.csv')
            sys.exit(0)
    p.wait()

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

Appendix B

Pourbaix Diagram Data Processing Code

```
import math
from matplotlib import lines
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

fileLocation = "location"


def interpolated_intercept(x, y1, y2):
    """Find the intercept of two curves, given by the same x data"""

    def intercept(point1, point2, point3, point4):
        """find the intersection between two lines
        the first line is defined by the line between point1 and point2
        the first line is defined by the line between point3 and point4
        each point is an (x,y) tuple.

        So, for example, you can find the intersection between
        intercept((0,0), (1,1), (0,1), (1,0)) = (0.5, 0.5)

        Returns: the intercept, in (x,y) format
        """

        def line(p1, p2):
            A = (p1[1] - p2[1])
            B = (p2[0] - p1[0])
            C = (p1[0]*p2[1] - p2[0]*p1[1])
            return A, B, -C

        def intersection(L1, L2):
            D  = L1[0] * L2[1] - L1[1] * L2[0]
            Dx = L1[2] * L2[1] - L1[1] * L2[2]
            Dy = L1[0] * L2[2] - L1[2] * L2[0]

            x = Dx / D
            y = Dy / D
            return x,y

        L1 = line([point1[0],point1[1]], [point2[0],point2[1]])
        L2 = line([point3[0],point3[1]], [point4[0],point4[1]])

        R = intersection(L1, L2)

        return R

    idx = np.argwhere(np.diff(np.sign(y1 - y2)) != 0)
    xc, yc = intercept((x[idx], y1[idx]),((x[idx+1], y1[idx+1])), ((x[idx],
y2[idx])), ((x[idx+1], y2[idx+1])))
    return xc,yc
```

```
concentration_lead_total = 10**(-4)
#pg 486
def calc_lead_pourbaix(pH):
    constant = .0592
    boundaries = 9.34
    HPbO2 = 10**(-4)
    PbO32 = 10**(-4)
    PbH2 = 10**(-4)
    Eqn_12_Boundary = (math.log10(concentration_lead_total)-12.65)/-2
    Eqn_13_Boundary = math.log10(HPbO2)+15.36
    Eqn_15_Boundary = (math.log10(PbO32)+31.32)/2

    concentration_H = 10 ** (-pH)
    H2_Eq = constant*pH
    O2_Eq = 1.223 - constant*pH
    Eq_16 = -.126 + .0295*math.log10(concentration_lead_total)
    Eq_21 = 1.449 - .1182*pH - .0295*math.log10(concentration_lead_total)
    Eq_7  = .248 - .0591*pH
    Eq_17 = .702 - (.0886*pH) + (.0295*math.log10(HPbO2))
    Eq_22 = .621-.0295*pH - .0295*math.log10(HPbO2)
    Eq_8 =  .972 - .0591*pH
    Eq_10 = 1.127 - .0591*pH
    Eq_18 = 2.094 - .2364*pH - .0886*math.log10(concentration_lead_total)
    Eq_19 = -.39 + (.0295*pH) - .0886*math.log10(HPbO2)
    Eq_24 = -1.507 - 0.0591*pH - 0.0295*math.log(PbH2)
    Eq_5 = 2.375-0.1773*pH
    Eq_6 = 1.547 - 0.0886*pH
    return(Eq_18, Eqn_12_Boundary, Eq_8, Eqn_13_Boundary, Eq_19, Eq_10,
Eq_21, Eq_16, Eq_7, Eq_17, Eq_24, Eq_5, Eq_6)

def plot_lead_pourbaix(E_in, ph_in):
    pH_range = []
    Eq_18 = []
    Eq_8 = []
    Eq_10 = []
    Eq_19 = []
    Eq_21 = []
    Eq_16 = []
    Eq_7 = []
    Eq_17 = []
    Eq_24 = []
    Eq_12_Boundary = []
    Eq_13_Boundary = []
    Eq_2_boundary = 3.84
    Eq_4 = 1.694
    Eq_5 = []
    Eq_6 = []
    Eq_1_boundary = 9.34
    pH_limit = 16
    for pH in range(0, pH_limit+1):
        pH_range.append(pH)
        pourbaix_tuple = calc_lead_pourbaix(pH)
        Eq_18.append(pourbaix_tuple[0])
        Eq_8.append(pourbaix_tuple[2])
        Eq_10.append(pourbaix_tuple[5])
        Eq_19.append(pourbaix_tuple[4])
        Eq_21.append(pourbaix_tuple[6])
```

```
        Eq_16.append(pourbaix_tuple[7])
        Eq_7.append(pourbaix_tuple[8])
        Eq_17.append(pourbaix_tuple[9])
        Eq_24.append(pourbaix_tuple[10])
        Eq_12_Boundary = pourbaix_tuple[1]
        Eq_13_Boundary = pourbaix_tuple[3]
        Eq_5.append(pourbaix_tuple[11])
        Eq_6.append(pourbaix_tuple[12])

    Eq_8_ph_range = [Eq_12_Boundary, Eq_13_Boundary]
    Eq_8_values = (calc_lead_pourbaix(Eq_12_Boundary)[2],
calc_lead_pourbaix(Eq_13_Boundary)[2])
    plt.plot(Eq_8_ph_range, Eq_8_values, label='8')

    Eq_19_ph_range = [Eq_13_Boundary, pH_limit]
    Eq_19_values = [calc_lead_pourbaix(Eq_13_Boundary)[4],
calc_lead_pourbaix(pH_limit)[4]]
    plt.plot(Eq_19_ph_range, Eq_19_values, label='19')

    xc1, yc1 =interpolated_intercept(np.array(pH_range), np.array(Eq_10),
np.array(Eq_18))
    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_10),
np.array(Eq_19))
    Eq_10_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_10_values =  [calc_lead_pourbaix(xc1[0][0])[5],
calc_lead_pourbaix(xc2[0][0])[5]]
    plt.plot(Eq_10_ph_range, Eq_10_values, label='10')

    PbO2 = [
        (xc1[0][0], yc1[0][0]),
        (xc2[0][0], yc2[0][0]),
        (16, Eq_19[16]),
        (16, 2),
        (0, 2)
    ]

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_21),
np.array(Eq_18))
    Eq_21_ph_range = [0, xc1[0][0]]
    Eq_21_values = [Eq_21[0], yc1[0][0]]
    plt.plot(Eq_21_ph_range, Eq_21_values, label='21')

    PbO2.append((xc1[0][0], yc1[0][0]))
    PbO2 = Polygon(PbO2)

    Eq_18_ph_range = [xc1[0][0], Eq_12_Boundary]
    Eq_18_Values = [yc1[0][0], calc_lead_pourbaix(Eq_12_Boundary)[0]]
    plt.plot(Eq_18_ph_range, Eq_18_Values, label='18')

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_16),
np.array(Eq_7))
    Eq_16_ph_range = [0, xc1[0][0]]
    Eq_16_values = [Eq_16[0], yc1[0][0]]
    plt.plot(Eq_16_ph_range, Eq_16_values, label='16')

    plt.vlines(x=Eq_12_Boundary, ymin=yc1[0][0],
ymax=calc_lead_pourbaix(Eq_12_Boundary)[2])
```

```
    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_17),
np.array(Eq_7))
    Eq_7_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_7_values = [calc_lead_pourbaix(xc1[0][0])[8],
calc_lead_pourbaix(xc2[0][0])[8]]
    plt.plot(Eq_7_ph_range, Eq_7_values, label='7')
    plt.vlines(x=Eq_13_Boundary, ymin=yc2[0][0],
ymax=calc_lead_pourbaix(Eq_13_Boundary)[4])

    Eq_17_ph_range = [xc2[0][0], pH_limit]
    Eq_17_values = [calc_lead_pourbaix(xc2[0][0])[9],
calc_lead_pourbaix(pH_limit)[9]]
    plt.plot(Eq_17_ph_range, Eq_17_values, label='17')

    Eq_24_ph_range = [0, 16]
    Eq_24_values = [Eq_24[0], Eq_24[16]]
    plt.plot(Eq_24_ph_range, Eq_24_values, label='24')

    Eq_4_ph_range = [0, Eq_2_boundary]
    plt.plot(Eq_4_ph_range, [Eq_4, Eq_4], '--', label='4')
    plt.vlines(x=Eq_2_boundary, ymin=Eq_4, ymax=2, linestyles='dashed')

    Pb_3plus = [
        (0, 2),
        (0, Eq_4),
        (Eq_2_boundary, Eq_4),
        (Eq_2_boundary, 2)
    ]
    Pb_3plus = Polygon(Pb_3plus)

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_5),
np.array(Eq_6))
    Eq_5_ph_range = [Eq_2_boundary, xc1[0][0]]
    Eq_5_values = [Eq_4, yc1[0][0]]
    plt.plot(Eq_5_ph_range, Eq_5_values, '--', label='5')

    Eq_6_ph_range = [xc1[0][0], 16]
    Eq_6_values = [yc1[0][0], Eq_6[16]]
    plt.plot(Eq_6_ph_range, Eq_6_values, '--', label='6')
    plt.vlines(x=Eq_1_boundary, ymin=-2, ymax=yc1[0][0], linestyles='dashed')

    PbO3_2minus = [
        (Eq_2_boundary, Eq_4),
        (xc1[0][0], yc1[0][0]),
        (16, Eq_6[16]),
        (16, 2),
        (Eq_2_boundary, 16)
    ]
    PbO3_2minus = Polygon(PbO3_2minus)

    yc2 = calc_lead_pourbaix(Eq_1_boundary)[10]
    Pb2plus = [
        (0, Eq_4),
        (Eq_2_boundary, Eq_4),
        (xc1[0][0], yc1[0][0]),
        (Eq_1_boundary, yc2),
```

```
    (0, Eq_24[0])
]
Pb2plus = Polygon(Pb2plus)

HPbO2_minus = [
    (xc1[0][0], yc1[0][0]),
    (16, Eq_6[16]),
    (16, Eq_24[16]),
    (Eq_1_boundary, yc2),
]
HPbO2_minus = Polygon(HPbO2_minus)

Pb3O4 = [
    (Eq_10_ph_range[0], Eq_10_values[0]),
    (Eq_10_ph_range[1], Eq_10_values[1]),
    (Eq_8_ph_range[1], Eq_8_values[1]),
    (Eq_8_ph_range[0], Eq_8_values[0])
]
Pb3O4 = Polygon(Pb3O4)

PbO = [
    (Eq_8_ph_range[0], Eq_8_values[0]),
    (Eq_8_ph_range[1], Eq_8_values[1]),
    (Eq_7_ph_range[1], Eq_7_values[1]),
    (Eq_7_ph_range[0], Eq_7_values[0]),
]
PbO = Polygon(PbO)

Pb = [
    (0, Eq_16[0]),
    (Eq_16_ph_range[0], Eq_16_values[1]),
    (Eq_17_ph_range[0], Eq_17_values[0]),
    (16, Eq_17[16]),
    (16, Eq_24[16]),
    (0, Eq_24[0])
]
Pb = Polygon(Pb)

PbH2 = [
    (0, Eq_24[0]),
    (16, Eq_24[16]),
    (0, -2)
]
PbH2 = Polygon(PbH2)

point = Point(E_in, ph_in)
solid = ''
aq = ''
gas = ''
if Pb_3plus.contains(point):
    aq = '(Pb)3+'
if PbO3_2minus.contains(point):
    aq = '(PbO3)2-'
if Pb2plus.contains(point):
    aq = '(Pb)2+'
if HPbO2_minus.contains(point):
    aq = '(HPbO2)-'
```

```
    if PbO2.contains(point):
        solid = 'PbO2'
    if PbO.contains(point):
        solid = 'PbO'
    if Pb3O4.contains(point):
        solid = 'Pb3O4'
    if Pb.contains(point):
        solid = 'Pb'
    if PbH2.contains(point):
        gas = 'PbH2'

    plt.legend(loc="lower left")
    plt.show()
    return (solid, aq, gas)



concentration_CU_total = .0001
#Units are mole/L
# pg 384
def calc_copper_pourbaix(pH):
    constant = .0592
    concentration_cu_plus = 10**(-6)
    concentration_cu_plus_plus = 10**(-6)
    concentration_HCuO2 = 10**(-6)
    concentration_CuO2  = 10**(-6)

    boundary_eq_11 = (math.log10(concentration_cu_plus_plus)-7.89)/(-2)
    boundary_eq_12 = math.log10(concentration_HCuO2) + 18.83
    boundary_eq_13 = (math.log10(concentration_CuO2)+31.98)/2
    boundary_eq_14 = .52 + (.0591*math.log10(concentration_cu_plus))
    boundary_eq_15 = .337 + (.0295*math.log10(concentration_cu_plus_plus))

    concentration_h = 10**(-pH)
    H2_eq = (-1)*constant*pH
    O2_eq = 1.223-(constant*pH)

    Eq_7 = .471 - (.0591*pH)
    Eq_9 = .669 - (.0591*pH)
    Eq_14 = boundary_eq_14
    Eq_15 = boundary_eq_15
    Eq_17 = 1.515-(.1182*pH)+(.0295*math.log10(concentration_CuO2))
    Eq_18 = .203 + (.0591*pH) + .0591*math.log10(concentration_cu_plus_plus)
    Eq_19 = 1.783 -(.1182*pH)+.0591*math.log10(concentration_HCuO2)
    Eq_20 = 2.56 - (.1773*pH) + .0591*math.log10(concentration_CuO2)
    Eq_5 = 1.733 - 0.1773*pH
    Eq_6 = 2.510 - 0.2364*pH
    return(Eq_18, boundary_eq_11, Eq_9, boundary_eq_12, Eq_17, Eq_7, Eq_15,
Eq_19, Eq_20, boundary_eq_13, Eq_5, Eq_6)

def plot_copper_pourbaix(E_in, ph_in):
    pH_range = []
    Eq_18 = []
    Eq_11_Boundary = []
    Eq_9 = []
    Eq_17 = []
    Eq_20 = []
```

```
    Eq_19 = []
    Eq_7 = []
    Eq_15 = []
    Eq_12_Boundary = []
    Eq_13_Boundary = []
    Eq_5 = []
    Eq_6 = []
    Eq_4 = 0.153
    Eq_1_boundary = 8.91
    Eq_3_boundary = 13.15

    pH_limit = 16
    for pH in range(0, pH_limit + 1):
        pH_range.append(pH)
        pourbaix_tuple = calc_copper_pourbaix(pH)
        Eq_18.append(pourbaix_tuple[0])
        Eq_9.append(pourbaix_tuple[2])
        Eq_17.append(pourbaix_tuple[4])
        Eq_7.append(pourbaix_tuple[5])
        Eq_15.append(pourbaix_tuple[6])
        Eq_19.append(pourbaix_tuple[7])
        Eq_20.append(pourbaix_tuple[8])
        Eq_11_Boundary = pourbaix_tuple[1]
        Eq_12_Boundary = pourbaix_tuple[3]
        Eq_13_Boundary = pourbaix_tuple[9]
        Eq_5.append(pourbaix_tuple[10])
        Eq_6.append(pourbaix_tuple[11])

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_18),
np.array(Eq_15))
    Eq_15_ph_range = [0, xc1[0][0]]
    Eq_15_values = [calc_copper_pourbaix(0)[6],
calc_copper_pourbaix(xc1[0][0])[6]]
    plt.plot(Eq_15_ph_range, Eq_15_values, label='Eqn 15')

    Cu_points = [
        (0, Eq_15[0]),
        (xc1[0][0], yc1[0][0])
    ]

    CutwoO_points = [
        (xc1[0][0], yc1[0][0])
    ]

    yc2 = calc_copper_pourbaix(Eq_11_Boundary)[0]
    Eq_18_ph_range = [xc1[0][0], Eq_11_Boundary]
    Eq_18_Values = [calc_copper_pourbaix(xc1[0][0])[0], yc2]
    plt.plot(Eq_18_ph_range, Eq_18_Values, label="Eqn 18")

    CuO_points = [
        (Eq_11_Boundary, yc2),
    ]

    CutwoO_points.append((Eq_11_Boundary, yc2))

    yc1 = calc_copper_pourbaix(Eq_12_Boundary)[2]
    Eq_9_ph_range = [Eq_11_Boundary, Eq_12_Boundary]
```

```
    Eq_9_Values = [calc_copper_pourbaix(Eq_11_Boundary)[2], yc1]
    plt.plot(Eq_9_ph_range, Eq_9_Values, label="Eqn 9")
    plt.vlines(x=Eq_12_Boundary, ymin=yc1, ymax=2)

    CuO_points.append((Eq_12_Boundary, yc1))
    CuO_points.append((Eq_12_Boundary, 2.0))
    CuO_points.append((Eq_11_Boundary, 2.0))

    CutwoO_points.append((Eq_12_Boundary, yc1))

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_19),
np.array(Eq_20))
    Eq_19_ph_range = [Eq_12_Boundary, xc1[0][0]]
    Eq_19_values = [calc_copper_pourbaix(Eq_12_Boundary)[7], yc1[0][0]]
    plt.plot(Eq_19_ph_range, Eq_19_values, label='Eqn 19')

    CutwoO_points.append((xc1[0][0], yc1[0][0]))

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_20),
np.array(Eq_7))
    Eq_20_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_20_values = [yc1[0][0], yc2[0][0]]
    plt.plot(Eq_20_ph_range, Eq_20_values, label='Eqn 20')

    CutwoO_points.append((xc2[0][0], yc2[0][0]))
    Cu_points.append((xc2[0][0], yc2[0][0]))

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_20),
np.array(Eq_17))
    Eq_17_ph_range = [xc1[0][0], 16]
    Eq_17_values = [yc1[0][0], Eq_17[16]]
    plt.plot(Eq_17_ph_range, Eq_17_values, label='Eqn 17')

    Cu_points.append((16, Eq_17[16]))
    Cu_points.append((16, -2))
    Cu_points.append((0, -2))

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_18),
np.array(Eq_7))
    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_17),
np.array(Eq_7))
    Eq_7_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_7_values = [calc_copper_pourbaix(xc1[0][0])[5],
calc_copper_pourbaix(xc2[0][0])[5]]
    plt.plot(Eq_7_ph_range, Eq_7_values, label='Eqn 7')

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_18),
np.array(Eq_9))
    plt.vlines(x=Eq_11_Boundary, ymin=yc1[0][0], ymax=2,
               label='Eq_11')


    plt.plot([0, Eq_1_boundary], [Eq_4, Eq_4], '--', label='Eqn 4')
    plt.vlines(x=Eq_1_boundary, ymin=Eq_4, ymax=2, linestyles='dashed')

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_5),
np.array(Eq_6))
```

```
    Eq_5_ph_range = [Eq_1_boundary, Eq_3_boundary]
    Eq_5_values = [Eq_4, yc2[0][0]]
    plt.plot(Eq_5_ph_range, Eq_5_values, '--',label='Eqn 5')

    Eq_6_ph_range = [xc2[0][0], 16]
    Eq_6_values = [yc2[0][0], Eq_6[16]]
    plt.plot(Eq_6_ph_range, Eq_6_values, '--',label='Eqn 6')
    plt.vlines(x=Eq_3_boundary, ymin=yc2[0][0], ymax=2, linestyles='dashed')

    CuPlus_points = [
        (0, Eq_4),
        (Eq_1_boundary, Eq_4),
        (xc2[0][0], yc2[0][0]),
        (16, Eq_6[16]),
        (16, -2),
        (0, -2)
    ]
    Cu_plus = Polygon(CuPlus_points)

    CuTwoplus_points = [
        (0, Eq_4),
        (Eq_1_boundary, Eq_4),
        (Eq_1_boundary, 2),
        (0, 2)
    ]
    CuTwoplus = Polygon(CuTwoplus_points)

    HCuO2_minus_points = [
        (Eq_1_boundary, Eq_4),
        (xc2[0][0], yc2[0][0]),
        (Eq_3_boundary, 2),
        (Eq_1_boundary, 2)
    ]
    HCuO2 = Polygon(HCuO2_minus_points)

    CuO2_two_minus_points = [
        (xc2[0][0], yc2[0][0]),
        (16, Eq_6[0][0]),
        (16, 2),
        (Eq_3_boundary, 2)
    ]
    CuO2_two_minus = Polygon(CuO2_two_minus_points)

    #Eq_20_ph_range = [xc1[0][0], xc2[0][0]]
    #Eq_20_values = [calc_copper_pourbaix(xc1[0][0])[8],
calc_copper_pourbaix(xc2[0][0])[8]]
    # plt.plot([0,16], [Eq_20[0], Eq_20[16]], label='Eqn 20')
    # plt.plot([0, 16], [Eq_19[0], Eq_19[16]], label='Eqn 19')
    # plt.plot([0, 16], [Eq_17[0], Eq_17[16]], label='Eqn 17')
    plt.legend(loc="lower left")
    plt.show()

    point = Point(E_in, ph_in)
    Cu = Polygon(Cu_points)
    CutwoO = Polygon(CutwoO_points)
    CuO = Polygon(CuO_points)
```

```
    solid = ''
    aq = ''

    if CuTwoplus.contains(point):
        aq = "(Cu)2+"
    if Cu_plus.contains(point):
        aq = "(Cu)+"
    if HCuO2.contains(point):
        aq = "(HCuO2)-"
    if CuO2_two_minus.contains(point):
        aq = "(CuO2)2-"
    if CuO.contains(point):
        solid = "CuO"
    if Cu.contains(point):
        solid = 'Cu'
    return (solid, aq)

# TODO Zinc Pourbaix Diagram
# pg 407
def calc_zinc_pourbaix(pH):
    concentration_Zn2 = 10**(-6)
    concentration_ZnO22 = 10**(-6)
    Eq_9 = -0.763 + 0.0295*math.log10(concentration_Zn2)
    Eq_5 = -0.439 - 0.0591*pH
    Eq_10 = 0.054 - 0.0886*pH + 0.0295*math.log10(concentration_ZnO22)
    Eq_11 = 0.441 - 0.1182*pH + 0.0295*math.log10(concentration_ZnO22)
    return (Eq_9, Eq_5, Eq_10, Eq_11)

def plot_zinc_pourbaix(E_in, pH_in):
    pH_range = []
    Eq_9 = []
    Eq_5 = []
    Eq_10 = []
    Eq_11 = []
    Eq_3_boundary = 9.21
    Eq_4_boundary = 13.11
    pH_limit = 16
    for pH in range(0, pH_limit+1):
        pH_range.append(pH)
        pourbaix_tuple = calc_zinc_pourbaix(pH)
        Eq_9.append(pourbaix_tuple[0])
        Eq_5.append(pourbaix_tuple[1])
        Eq_10.append(pourbaix_tuple[2])
        Eq_11.append(pourbaix_tuple[3])

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_9),
np.array(Eq_5))
    Eq_9_ph_range = [0, xc1[0][0]]
    Eq_9_values = [Eq_9[0], yc1[0][0]]
    plt.plot(Eq_9_ph_range, Eq_9_values, label='Eqn 9')
    plt.vlines(x=xc1[0][0], ymin=yc1[0][0], ymax=2)

    Zn2plus = [
        (0, -2),
        (Eq_3_boundary, -2),
        (Eq_3_boundary, 2),
        (0, 2.0),
```

```
    ]
    Zn2plus = Polygon(Zn2plus)

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_5),
np.array(Eq_10))
    Eq_5_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_5_values = [yc1[0][0], yc2[0][0]]
    plt.plot(Eq_5_ph_range, Eq_5_values, label='Eqn 5')
    plt.vlines(x=xc2[0][0], ymin=yc2[0][0], ymax=2)

    ZnOHtwo = [
        (xc1[0][0], yc1[0][0]),
        (xc2[0][0], yc2[0][0]),
        (xc2[0][0], 2.0),
        (xc1[0][0], 2.0),
    ]
    ZnOHtwo = Polygon(ZnOHtwo)

    Zn = [
        (16, Eq_11[16]),
        (16, -2),
        (0, -2),
        (0, Eq_9[0]),
        (xc1[0][0], yc1[0][0]),
        (xc2[0][0], yc2[0][0]),
    ]

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_10),
np.array(Eq_11))
    Eq_10_ph_range = [xc2[0][0], xc1[0][0]]
    Eq_10_values = [yc2[0][0], yc1[0][0]]
    plt.plot(Eq_10_ph_range, Eq_10_values, label='Eqn 10')
    # plt.vlines(x=xc1[0][0], ymin=yc1[0][0], ymax=2)
    Zn.append((xc1[0][0], yc1[0][0]))
    Zn = Polygon(Zn)

    HZnOtwo_minus = [
        (Eq_3_boundary, -2),
        (Eq_4_boundary, -2),
        (Eq_4_boundary, 2),
        (Eq_3_boundary, 2),
    ]
    HZnOtwo_minus = Polygon(HZnOtwo_minus)

    Eq_11_ph_range = [xc1[0][0], 16]
    Eq_11_values = [yc1[0][0], Eq_11[16]]
    plt.plot(Eq_11_ph_range, Eq_11_values, label='Eqn 11')
    plt.vlines(x=xc2[0][0], ymin=yc2[0][0], ymax=2)
    plt.vlines(x=Eq_3_boundary, ymin=-2, ymax=2, linestyles='dashed')
    plt.vlines(x=Eq_4_boundary, ymin=-2, ymax=2, linestyles='dashed')

    ZnOtwo_minus = [
        (Eq_4_boundary, -2),
        (16, -2),
        (16, 2.0),
        (Eq_4_boundary, 2.0),
    ]
```

```
    ZnOtwo_minus = Polygon(ZnOtwo_minus)

    # plt.plot([0,16], [Eq_1[0], Eq_1[16]], label='Eqn 1')
    # plt.plot([0,16], [Eq_2[0], Eq_2[16]], label='Eqn 2')
    # plt.plot([0,16], [Eq_3[0], Eq_3[16]], label='Eqn 3')
    # plt.plot([0,16], [Eq_4[0], Eq_4[16]], label='Eqn 4')
    plt.legend(loc="lower left")
    plt.show()

    point = Point(E_in, pH_in)

    solid = ''
    aq = ''

    if Zn2plus.contains(point):
        aq = "(Zn)2+"
    if HZnOtwo_minus.contains(point):
        aq = "(HZnO2)-"
    if ZnOtwo_minus.contains(point):
        aq = "(ZnO2)2-"
    if Zn.contains(point):
        solid = "Zn"
    if ZnOHtwo.contains(point):
        solid = "Zn(OH)2"
    return (solid, aq)

# TODO Iron Pourbaix Diagram
# pg 307
def calc_iron_pourbaix(pH):
    concentration_fe2_plus = 10**(-6)
    concentration_fe3_plus = 10**(-6)
    concentration_HFeO2_minus = 10**(-6)

    Eq_23 = -0.44 + (0.0295)*math.log10(concentration_fe2_plus)
    Eq_4 = 0.771 +
(0.0591)*math.log10(concentration_fe3_plus/concentration_fe2_plus)
    # Eq_20 = ("Fe3+") = 10**(-0.72-3*pH)
    Eq_28 = 0.728-(0.1773*pH)-(0.0591*math.log10(concentration_fe2_plus))
    Eq_17 = 0.221 - 0.0591*pH
    Eq_13 = -0.085 - 0.0591*pH
    Eq_26 = 0.98 - 0.2364*pH - (0.0886*math.log10(concentration_fe2_plus))
    Eq_24 = 0.493 - 0.0886*pH + 0.0295*math.log10(concentration_HFeO2_minus)
    Eq_27 = -1.819 + 0.0295*pH - 0.0886
*math.log10(concentration_HFeO2_minus)
    Eq_9 = 1.700 - 0.1580*pH
    Eq_10 = 1.652 - 0.1379*pH
    Eq_5 = 0.914 - 0.0591*pH
    Eq_11 = 1.559 - 0.1182*pH
    Eq_6 = 1.191 - 0.1182*pH
    Eq_7 = -0.675 + 0.0591*pH
    Eq_8 = 1.001 - 0.0738*pH
    HYDROGEN_EQ = -0.0592*pH
    OXYGEN_EQ = 1.223-(0.0592*pH)

    return (Eq_23, Eq_4, Eq_28, Eq_17, Eq_13, Eq_26, Eq_24, Eq_27, Eq_9,
Eq_10, Eq_5, Eq_11, Eq_6, Eq_7, Eq_8, HYDROGEN_EQ, OXYGEN_EQ)
```

```
def plot_iron_pourbaix(E_in, ph_in):
    pH_range = []
    Eq_23 = []
    Eq_4 = []
    Eq_20 = []
    Eq_28 = []
    Eq_17 = []
    Eq_13 = []
    Eq_26 = []
    Eq_24 = []
    Eq_27 = []
    Eq_9 = []
    Eq_10 = []
    Eq_5 = []
    Eq_11 = []
    Eq_6 = []
    Eq_7 = []
    Eq_8 = []
    Hydrogen = []
    Oxygen = []
    Eq_1_boundary = 10.53
    Eq_2_boundary = 2.43
    Eq_3_boundary = 4.69
    pH_limit = 16
    for pH in range(0, pH_limit+1):
        pH_range.append(pH)
        pourbaix_tuple = calc_iron_pourbaix(pH)
        Eq_23.append(pourbaix_tuple[0])
        Eq_4.append(pourbaix_tuple[1])
        # Eq_20.append(pourbaix_tuple[2])
        Eq_28.append(pourbaix_tuple[2])
        Eq_17.append(pourbaix_tuple[3])
        Eq_13.append(pourbaix_tuple[4])
        Eq_26.append(pourbaix_tuple[5])
        Eq_24.append(pourbaix_tuple[6])
        Eq_27.append(pourbaix_tuple[7])
        Eq_9.append(pourbaix_tuple[8])
        Eq_10.append(pourbaix_tuple[9])
        Eq_5.append(pourbaix_tuple[10])
        Eq_11.append(pourbaix_tuple[11])
        Eq_6.append(pourbaix_tuple[12])
        Eq_7.append(pourbaix_tuple[13])
        Eq_8.append(pourbaix_tuple[14])
        Hydrogen.append(pourbaix_tuple[15])
        Oxygen.append(pourbaix_tuple[16])

    Eq_2_values = calc_iron_pourbaix(Eq_2_boundary)
    Eq_9_ph_range = [0, Eq_2_boundary]
    Eq_9_values = [Eq_9[0], Eq_2_values[8]]
    plt.plot(Eq_9_ph_range, Eq_9_values, '--', label='Eqn 9')

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_4),
np.array(Eq_28))
    Eq_4_ph_range = [0, Eq_2_boundary]
    Eq_4_values = [Eq_4[0], Eq_2_values[1]]
    plt.plot(Eq_4_ph_range, Eq_4_values, '--', label='Eqn 4')
    plt.vlines(x=xc1[0][0], ymin=yc1[0][0], ymax=2)
```

```
    plt.vlines(x=Eq_2_boundary, ymin=Eq_2_values[1], ymax=Eq_2_values[8],
linestyles='dashed')

    Fe2_O3 = [
        (xc1[0][0], yc1[0][0]),
        (xc1[0][0], 2),
        (16, 2),
        (16, Eq_17[16]),
    ]

    Fe_3_plus = [
        (0, Eq_9[0]),
        (Eq_2_boundary, Eq_2_values[8]),
        (Eq_2_boundary, Eq_2_values[1]),
        (0, Eq_4[0])
    ]

    Fe_3_plus = Polygon(Fe_3_plus)

    Eq_3_values = calc_iron_pourbaix(Eq_3_boundary)
    Eq_10_ph_range = [Eq_2_boundary, Eq_3_boundary]
    Eq_10_values = [Eq_2_values[9], Eq_3_values[9]]
    plt.plot(Eq_10_ph_range, Eq_10_values, '--', label='Eqn 10')

    Eq_5_ph_range = [Eq_2_boundary, Eq_3_boundary]
    Eq_5_values = [Eq_2_values[10], Eq_3_values[10]]
    plt.plot(Eq_5_ph_range, Eq_5_values, '--', label='Eqn 5')
    plt.vlines(x=Eq_3_boundary, ymin=Eq_3_values[10], ymax=Eq_3_values[9],
linestyles='dashed')

    FeOH_2_plus = [
        (Eq_2_boundary, Eq_2_values[9]),
        (Eq_3_boundary, Eq_3_values[9]),
        (Eq_3_boundary, Eq_3_values[10]),
        (Eq_2_boundary, Eq_2_values[10])
    ]

    FeOH_2_plus = Polygon(FeOH_2_plus)

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_26),
np.array(Eq_28))
    Eq_28_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_28_values = [yc1[0][0], yc2[0][0]]
    plt.plot(Eq_28_ph_range, Eq_28_values, label='Eqn 28')

    Fe2_O3.append((xc2[0][0], yc2[0][0]))
    Fe2_O3 = Polygon(Fe2_O3)

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_23),
np.array(Eq_26))
    Eq_26_ph_range = [xc2[0][0], xc1[0][0]]
    Eq_26_values = [yc2[0][0], yc1[0][0]]
    plt.plot(Eq_26_ph_range, Eq_26_values, label='Eqn 26')

    Eq_23_ph_range = [0, xc1[0][0]]
    Eq_23_values = [Eq_23[0], yc1[0][0]]
    plt.plot(Eq_23_ph_range, Eq_23_values, label='Eqn 23')
```

```
    Eq_17_ph_range = [xc2[0][0], 16]
    Eq_17_values = [yc2[0][0], Eq_17[16]]
    plt.plot(Eq_17_ph_range, Eq_17_values, label='Eqn 17')

    Fe3_O4 = [
        (xc1[0][0], xc1[0][0]),
        (xc2[0][0], yc2[0][0]),
        (16, Eq_17[16]),
        (16, Eq_13[16])
    ]

    Fe3_O4 = Polygon(Fe3_O4)

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_13),
np.array(Eq_24))
    Eq_13_ph_range = [xc1[0][0], xc2[0][0]]
    Eq_13_values = [yc1[0][0], yc2[0][0]]
    plt.plot(Eq_13_ph_range, Eq_13_values, label='Eqn 13')

    Fe = [
        (0, Eq_23[0]),
        (xc1[0][0], yc1[0][0]),
        (16, Eq_13[16]),
        (16, -2),
        (0, -2)
    ]

    Fe = Polygon(Fe)

    Eq_24_ph_range = [xc2[0][0], 16]
    Eq_24_values = [yc2[0][0], Eq_24[16]]
    plt.plot(Eq_24_ph_range, Eq_24_values, label='Eqn 24')

    Eq_27_ph_range = [xc2[0][0], 16]
    Eq_27_values = [yc2[0][0], Eq_27[16]]
    plt.plot(Eq_27_ph_range, Eq_27_values, label='Eqn 27')

    xc1, yc1 = interpolated_intercept(np.array(pH_range), np.array(Eq_11),
np.array(Eq_7))
    Eq_11_ph_range = [Eq_3_boundary, xc1[0][0]]
    Eq_11_values = [Eq_3_values[11], yc1[0][0]]
    plt.plot(Eq_11_ph_range, Eq_11_values, '--', label='Eqn 11')

    xc2, yc2 = interpolated_intercept(np.array(pH_range), np.array(Eq_6),
np.array(Eq_7))
    Eq_6_ph_range = [Eq_3_boundary, xc2[0][0]]
    Eq_6_values = [Eq_3_values[12], yc2[0][0]]
    plt.plot(Eq_6_ph_range, Eq_6_values, '--', label='Eqn 6')
    plt.vlines(x=Eq_1_boundary, ymin=-2,
ymax=calc_iron_pourbaix(Eq_1_boundary)[12], linestyles='dashed')

    Eq_7_ph_range = [xc2[0][0], xc1[0][0]]
    Eq_7_values = [yc2[0][0], yc1[0][0]]
    plt.plot(Eq_7_ph_range, Eq_7_values, '--', label='Eqn 7')

    Fe_2_plus = [
```

```
        (0, Eq_4[0]),
        (Eq_2_boundary, Eq_2_values[10]),
        (Eq_3_boundary, Eq_3_values[10]),
        (xc2[0][0], yc2[0][0]),
        (Eq_1_boundary, -2),
        (0, -2)
    ]
    Fe_2_plus = Polygon(Fe_2_plus)

    Eq_8_ph_range = [xc1[0][0], 16]
    Eq_8_values = [yc1[0][0], Eq_8[16]]
    plt.plot(Eq_8_ph_range, Eq_8_values, '--', label='Eqn 8')

    FeO4_2_minus = [
        (0, Eq_9[0]),
        (Eq_2_boundary, Eq_2_values[10]),
        (Eq_3_boundary, Eq_3_values[10]),
        (xc1[0][0], yc1[0][0]),
        (16, Eq_8[16]),
        (16, 2),
        (0, 2),
    ]

    FeO4_2_minus = Polygon(FeO4_2_minus)

    Fe_OH2_plus = [
        (Eq_3_boundary, Eq_3_values[11]),
        (xc1[0][0], yc1[0][0]),
        (xc2[0][0], yc2[0][0]),
        (Eq_3_boundary, Eq_3_values[12])
    ]

    Fe_OH2_plus = Polygon(Fe_OH2_plus)

    HFeO2 = [
        (Eq_1_boundary, -2),
        (xc2[0][0], yc2[0][0]),
        (xc1[0][0], yc1[0][0]),
        (16, Eq_8[16]),
        (16, -2)
    ]

    HFeO2 = Polygon(HFeO2)

    plt.plot([0,16], [Hydrogen[0], Hydrogen[16]], 'k', label='Hydrogen
Equilibrium')
    plt.plot([0,16], [Oxygen[0], Oxygen[16]], 'k', label='Oxygen
Equilibrium')
    plt.legend(loc="lower left")
    plt.show()

    point = Point(E_in, ph_in)
    solid = ''
    aq = ''
    if FeO4_2_minus.contains(point):
        aq = '(FeO4)2-'
    if Fe_3_plus.contains(point):
```

```
        aq = '(Fe)3+'
    if FeOH_2_plus.contains(point):
        aq = '(FeOH)2+'
    if Fe_OH2_plus.contains(point):
        aq = '(Fe(OH)2)+'
    if Fe_2_plus.contains(point):
        aq = '(Fe)2+'
    if HFeO2.contains(point):
        aq = 'HFeO2'
    if Fe2_O3.contains(point):
        solid = 'Fe2O3'
    if Fe3_O4.contains(point):
        solid = 'Fe3O4'
    if Fe.contains(point):
        solid = 'Fe'
    return (solid, aq)

def calc_total_alkalinity(pH):
    PCO2 = 10**(-3.5)
    k1 = 10**(-1.5)
    k2 = 10**(-6.3)
    k3 = 10**(-10.3)
    concentration_H = 10**(-pH)
    concentration_OH = 10**(-1*(14-pH))
    concentration_H2CO3 = PCO2*k1
    concentration_HCO3 = k2*concentration_H2CO3/concentration_H
    concentration_CO32 = k3 *concentration_HCO3/concentration_H

    alkalinity = concentration_HCO3 + 2*concentration_CO32 + concentration_OH
+ concentration_H
    return alkalinity

def calc_copper_complexation(pH):
    Ks0 = 10**(8.7)
    Ks1 = 10
    Ks2 = 10**(-7.5)
    Ks3 = 10**(-30.9)

    log_concentration_Cu2 = math.log10(Ks0)-2*pH
    log_concentration_CuOH = math.log10(Ks1)-pH
    log_concentration_CuOH2 = math.log10(Ks2)
    log_concentration_CuOH42 = math.log10(Ks3) +pH
    return(log_concentration_Cu2, log_concentration_CuOH,
log_concentration_CuOH2, log_concentration_CuOH42)

def calc_HOCL(pH):
    Ka = 3.5*(10**(-8))
    totA = 10**(-4)
    concentration_H= 10**(-pH)
    concentration_HOCL = totA*(1/(1+Ka/concentration_H))
    concentration_OCL = totA*(1/(1+(concentration_H/Ka)))
    log_concentration_OCL = math.log10(concentration_OCL)
    log_concentration_HOCL = math.log10(concentration_HOCL)
    return(log_concentration_HOCL, log_concentration_OCL)

def calc_hardness(pH):
    #caTotal = 10**(-3)
```

```
    #PCO2 =  10**(-3.5)
    #H2CO3 = 10**(-5)
    log_concentration_HCO3 = pH-11.3
    log_concentration_CO32 = 2*pH-21.6
    log_concentration_Ca2 = 13.3-(2*pH)
    return(log_concentration_HCO3, log_concentration_CO32,
log_concentration_Ca2)

# total dissolved solids
def calc_tds(ec):
    # assume constant as 0.64
    return (0.64)*ec

def calc_ionic_strength(tds):
    return (2.5*(10**-5))*(tds)

def graphComputeAll():
    i =1

def graphAll():
    i =1

def main():
    # print(plot_copper_pourbaix(3, 1.7))
    plot_lead_pourbaix()
    # print(plot_zinc_pourbaix(1, 3))
    # plot_iron_pourbaix()

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

Appendix C

Procedure For Configuring the Raspberry Pi for Data Collection

1. To begin, download the necessary operating system onto raspberry pi, can be done on an external computer and requires a 64-gigabyte microSD card as well as a microSD card adapter
2. Visit raspberry pi website to download the operating system at: https://www.raspberrypi.com/software/
3. First you will need to download the Raspberry Pi Imager software. Follow the instructions on the page to download for Mac or Windows
4. Open the Raspberry Pi Imager from your applications folder
5. When the imager opens select the Raspberry Pi OS (32bit)
6. Plug in microSD card adapter into computer and select the microSD card listed after clicking storage button
7. Select write and the raspberry pi imager will show the amount of progress completed throughout the entire writing process
8. Once the microSD card has finished writing and verifying, eject the microSD card adapter and remove
9. Now that the microSD card is configured, continue by taking the Raspberry Pi baseboard and locating the microSD card slot located on the bottom side of the board
10. Insert the microSD card into the slot
11. Plug in all auxiliary hardware to connect the mouse, keyboard, and LCD screen. This may be performed in any order but should consist of the following connections:
    1. Connect the mouse USB cord to USB2 or USB3 port Raspberry Pi
    2. Connect the keyboard USB2 or USB3 port on Raspberry Pi
    3. Connect mini-HDMI to USB cord to the DC Touch slot along the left side of the LCD screen and a USB2 or USB3 port on Raspberry Pi, respectively
    4. Connect HDMI to micro-HDMI cord in HDMI port along the left side of the LCD screen and to the micro-HDMI port on the Raspberry Pi
12. After this, the Whitebox labs carrier boards should be outfitted with the circuits for each sensor. There are two electrically isolated slots and one non-isolated slot on each carrier board. The circuits each have three pins that slide into these slots on the carrier board. Ensure that the text on the circuit and the text on the carrier board are both reading right side up and carefully slide the circuits into a slot. All circuits should be on electrically isolated slots, excluding the temperature or flow circuits.
13. The next step is to attach the Whitebox labs carrier boards to the Raspberry Pi, all the boards contain GPIO pins along the top side. Align these pins directly over the GPIO pins on the Raspberry Pi and gently press the two components together until the pins are stacked and securely attached
14. The sensors may now be connected to the system. First add an SMA to BNC adapter to each circuit at the end of each sensor cable. The adapters screw directly onto the male SMA end of each cable. Locate the white BNC connectors that are in front of the circuits and connect each sensor to its respective circuit through the BNC connector.
15. Now the Raspberry Pi must be powered on use the 5V USB-C power supply cord to plug in the Raspberry Pi from a power outlet. There are two USB-C slots on the Raspberry Pi,

ensure that the cord is plugged into the leftmost port, if your power supply cord has a switch included, you may switch it on at this point

16. Allow 10-30 seconds for the Raspberry Pi to boot up, note that if you see any green granules/lines on the LCD screen, you should double check all cable connections
17. Once the Raspberry Pi has booted up for the first time Setup wizard will walk you through the basic start-up and prompt you for localization, a new password, Wi-Fi network, etc.
18. To download the code to a new device run the command $ git clone https://github.com/SmartWaterFilter/AtlasScientificDataAcqusition.git
19. This command should ask for a username and password which are
    1. Username: SmartWaterFilter and
    2. Password: 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
20. This command will clone the Github repository, after which you can enter the command $ ./start, which should begin polling values from the sensors after prompting for user inputs on comments, plotting etc. The user may decide if they would like to add any comments or have the data plotted during the data collection
21. After completing these steps, the sensors may be inserted into whatever sample is to be measured, if they are not already in solution or in-line
22. To stop the polling of data simply press ctrl-c and the data acquisition code will stop. The user will be asked for several other inputs including the username and password before the data is automatically pushed to the online repository
23. Steps 19-23 may be repeated for any number of runs
24. When the desired data acquisition is complete, the terminal window may be exited, and the Raspberry Pi can be shut down

References

Adu-Manu, K. S., Katsriku, F. A., Abdulai, J. D., & Engmann, F. (2020). Smart River Monitoring Using Wireless Sensor Networks. *Wireless Communications and Mobile Computing*, *2020*. https://doi.org/10.1155/2020/8897126

Bergendahl, J. A., & Stevens, L. (2005). Oxidation reduction potential as a measure of disinfection effectiveness for chlorination of wastewater. *Environmental Progress*, *24*(2), 214–222. https://doi.org/10.1002/ep.10074

Drinking Water Standards and Regulations | Public Water Systems | Drinking Water | Healthy Water | CDC. (n.d.). Retrieved November 23, 2021, from https://www.cdc.gov/healthywater/drinking/public/regulations.html

Fathi, E., Zamani-Ahmadmahmoodi, R., & Zare-Bidaki, R. (2018). Water quality evaluation using water quality index and multivariate methods, Beheshtabad River, Iran. *Applied Water Science*, *8*(7). https://doi.org/10.1007/s13201-018-0859-7

Geetha, S., & Gouthami, S. (2016). Internet of things enabled real time water quality monitoring system. *Smart Water*, *2*(1). https://doi.org/10.1186/s40713-017-0005-y

Jung, H., Kim, U., Seo, G., Lee, H., & Lee, C. (2009). Effect of Dissolved Oxygen (DO) on Internal Corrosion of Water Pipes. In *Korean Society of Environmental Engineers* (Vol. 14, Issue 3).

Kim, H., Kwon, S., Han, S., Yu, M., Kim, J., Gong, S., & Colosimo, M. F. (2006). New ORP/pH-based control strategy for chlorination and dechlorination of wastewater: Pilot scale application. *Water Science and Technology*, *53*(6), 145–151. https://doi.org/10.2166/wst.2006.188

Lambrou, T. P., Anastasiou, C. C., Panayiotou, C. G., & Polycarpou, M. M. (2014). A low-cost sensor network for real-time monitoring and contamination detection in drinking water distribution systems. *IEEE Sensors Journal*, *14*(8), 2765–2772. https://doi.org/10.1109/JSEN.2014.2316414

Leventeli, Y., & Yalcin, F. (2021). Data analysis of heavy metal content in riverwater: multivariate statistical analysis and inequality expressions. *Journal of Inequalities and Applications*, *2021*(1). https://doi.org/10.1186/s13660-021-02549-3

Lockridge, G., Dzwonkowski, B., Nelson, R., & Powers, S. (2016). Development of a low-cost arduino-based sonde for coastal applications. *Sensors (Switzerland)*, *16*(4). https://doi.org/10.3390/s16040528

Parra, L., Sendra, S., García, L., & Lloret, J. (2018). Design and deployment of low-cost sensors for monitoring the water quality and fish behavior in aquaculture tanks during the feeding process. *Sensors (Switzerland)*, *18*(3). https://doi.org/10.3390/s18030750

Pasika, S., & Gandla, S. T. (2020). Smart water quality monitoring system with cost-effective using IoT. *Heliyon*, *6*(7). https://doi.org/10.1016/j.heliyon.2020.e04096

Pierce, G., Gonzalez, S. R., Roquemore, P., & Ferdman, R. (2019). Sources of and solutions to mistrust of tap water originating between treatment and the tap: Lessons from Los Angeles County. *The Science of the Total Environment*, *694*. https://doi.org/10.1016/J.SCITOTENV.2019.133646

Suslow, T. v. (n.d.). *Oxidation-Reduction Potential (ORP) for Water Disinfection Monitoring, Control, and Documentation PUBLICATION 8149 UNIVERSITY OF CALIFORNIA Division of Agriculture and Natural Resources*. http://anrcatalog.ucdavis.edu

Vijayakumar, N., & Ramya, R. (2013). The Real Time Monitoring of Water Quality in IoT Environment. In *International Journal of Science and Research* (Vol. 4). www.ijsr.net

Wu, J., Cao, M., Tong, D., Finkelstein, Z., & Hoek, E. M. v. (n.d.). *A critical review of point-of-use drinking water treatment in the United States*. https://doi.org/10.1038/s41545-021-00128-z

Zennaro, M., Floros, A., Dogan, G., Sun, T., Cao, Z., Huang, C., Bahader, M., Ntareme, H. ', & Bagula, A. (n.d.). *On the design of a Water Quality Wireless Sensor Network (WQWSN): an Application to Water Quality Monitoring in Malawi*.

Zhang, D., Sullivan, T., Briciu-Burghina, C., Murphy, K., Mcguinness, K., O'connor, N. E., Smeaton, A., & Regan, F. (n.d.). *Detection and Classification of Anomalous Events in Water Quality Datasets Within a Smart City-Smart Bay Project*.