Eco Sensor Pod

By Austin Zuffi and Michael Chamoures

Figure 1: Physical Buoy System

Senior Project

Computer Engineering Department

California Polytechnic State University

San Luis Obispo

June 2013

© 2013 Austin Zuffi, Michael Chamoures
# Table of Contents

Acknowledgements ................................................................................................................. 4  
List of Tables ............................................................................................................................. 5  
List of Figures ............................................................................................................................ 6  
Abstract ..................................................................................................................................... 7  
Introduction ............................................................................................................................... 8  
Background ............................................................................................................................... 10  
  Hardware ................................................................................................................................. 10  
  Software ................................................................................................................................. 10  
Requirements ............................................................................................................................. 10  
  Hardware ................................................................................................................................. 10  
  Software ................................................................................................................................. 10  
Specifications ............................................................................................................................. 12  
  Hardware ................................................................................................................................. 12  
  Software ................................................................................................................................. 12  
Design ....................................................................................................................................... 13  
  Hardware ................................................................................................................................. 13  
    Prototype (Winter Quarter, 2013) ....................................................................................... 13  
    PCB (Spring Quarter, 2013) .............................................................................................. 14  
  Software .................................................................................................................................. 16  
Testing/Verification .................................................................................................................... 21  
  Hardware ................................................................................................................................. 21  
  Software ................................................................................................................................. 22  
Conclusion ................................................................................................................................. 24  
  Hardware ................................................................................................................................. 24  
  Software ................................................................................................................................. 24  
Bibliography ............................................................................................................................... 26  
Appendices ............................................................................................................................... 28  
 A - Senior Project Analysis .................................................................................................... 28  
    Summary of Functional Requirements ............................................................................. 28  
  Hardware ................................................................................................................................. 28  
  Software ................................................................................................................................. 28  
    Primary Constraints ............................................................................................................ 28  
    Economic .............................................................................................................................. 29  
    Environmental .................................................................................................................... 29  
    Manufacturability ............................................................................................................... 29  
    Sustainability ...................................................................................................................... 29  
    Ethical ................................................................................................................................. 30  
    Health and Safety .............................................................................................................. 30  
    Social and Political ............................................................................................................. 30  
    Development ...................................................................................................................... 30
Acknowledgements

This project was made possible thanks to the help of our advisor, Bridget Benson, who was able to connect us to our client, Peter Shin. With the support of them, the various references they provided, some current class material, and good old-fashioned research we were able to learn and develop a solution to the problem they presented to us.
List of Tables

Table 1: Appropriate Component Voltages ........................................................................21
List of Figures

Figure 1: Physical Buoy System ........................................................................................................ 1
Figure 3: UCSD’s Prototype ................................................................................................................ 8
Figure 4: UCSD Team’s Schematic .................................................................................................... 12
Figure 5: First Quarter Prototype ..................................................................................................... 14
Figure 6: Diptrace PCB Layout ........................................................................................................ 15
Figure 7: The Final Product ................................................................................................................ 16
Figure 8: Real-Time Data Viewer (RDV) - The original charting tool ............................................ 17
Figure 9: Eco Sensor Pod Web Application ...................................................................................... 18
Figure 10: Front-end to Middle-end Program Flow Diagram ............................................................ 19
Figure 11: DataFacilitator Middle-End Program Flow Diagram ..................................................... 19
Figure 13: Eco Sensor Pod Web App Output .................................................................................... 23
Figure 14: Original RDV Output ........................................................................................................ 24
Figure 15: Eco Sensor Pod Web App Home Page ........................................................................... 35
Figure 16: Example snapshot of Eco Sensor Pod Web App "About" Page ........................................ 35
Figure 17: Host Address and Port Number Input Forms ................................................................. 36
Figure 18: Duration Selection Dropdown ......................................................................................... 36
Figure 19: Real-Time Form Option .................................................................................................. 36
Figure 20: End Date and Time Input Forms ...................................................................................... 36
Figure 21: Example of Channel Name List Links ............................................................................ 37
Figure 22: Example Output from Web App ..................................................................................... 37
Figure 23: Real-Time Update Control Options ................................................................................ 38
Abstract

The EcoSensor Pod, designed by the UCSD Data Turbine group, is a self-contained hardware/software solution for environmental monitoring, disaster response, and other real-time sensing applications. We improved upon the design of the Sensor Pod by 1. Developing a custom PCB for the system to remove the problem of wires becoming de-attached in harsh environment deployments and 2. Creating an easy-to-use web application that provides the UCSD team with the ability to view various charts displaying both past and live, real-time data. These modifications were successfully implemented and tested and will make future deployments of the EcoSensor Pod more robust and user friendly.

Figure 2: System Block Diagram
Introduction

The EcoSensor Pod, designed by the UCSD Data Turbine group, is a self-contained hardware/software solution for environmental monitoring, disaster response, and other real-time sensing applications. We recreated UCSD’s current buoy sensor pod in a CAD program, because a PCB is more easily deployed than a hardwired system. The team places this in a waterproof buoy. Various sensors plug in via RS232 ports and hang into the water. An Android device plugs in via a IOIO board [6] and sends the sensor data back to UCSD’s DataTurbine server [4].

We also created an easy-to-use web application that provides the UCSD team with the ability to view various charts displaying both past and live, real-time data. This data comes from their deployed Sensor Pods for applications in environmental monitoring, disaster response, and other real-time sensor applications. This is made possible by using Apache Tomcat to power a Java servlet that fetches data from the DataTurbine server upon request from the web application. The servlet then forwards the data to the web application publicly hosted on an Amazon cloud (AWS) to be charted via the HighStock charting API.

After multiple deployments, the UCSD Data Turbine group decided it would be useful to make hardware and software modifications to the sensor pod to make it more robust and user friendly.

The goal of the hardware redesign was to replace the current hardwired system (Figure 3) to make it much cleaner, more reliable, and more easily assembled. The current system might become disconnected under harsh weather conditions. Also, assembling a new buoy involves careful wiring and adherence to the schematic. This wastes time and poses a high chance for error. These challenges are overcome by redesigning the system as a printed circuit board.

Figure 3: UCSD's Prototype
The problem with the current software of the system is that a local installation of the client program used to display the data from the sensor system is required and does not offer the option to continuously update the charts with streaming, real-time data. In an effort to provide the DataTurbine group with a more robust and easily accessible mechanism of viewing data, we designed a new publicly hosted web application that they can access and use from any web browser with an internet connection. The web application will make use of various open-source APIs combined together to provide the team all the functionalities of the past viewing software, as well as the new real-time updating option.
Background

Hardware
The board redesign required computer aided design of a PCB schematic and layout, physical board fabrication, and soldering. The CPE curriculum provided this education in IME 156: Basic Electronics Manufacturing. Austin secured this knowledge when he repeated the course as a teacher’s assistant. This also established a solid relationship with the professor, Gary Perks, who would later serve as a resource for quick solutions. Lastly, this provided the team with unlimited access to the IME lab, including access to a complete set of hand tools, soldering stations, and a photolithography room used for board fabrication.

Software
First, it was necessary to learn how the open source DataTurbine middleware project/API functioned in order to learn how data is communicated between the source, server, and sink client. Next, we experimented with their current sink client program (RDV) to get an idea of the expected functionality requirements for the web application. When it was time to design the project, it was also necessary to understand the organization and common design of a Java Web Application. From there, we learned how to setup a Java Web Application in Eclipse, import different JAR files, and connect to a Tomcat server. We then found out how to connect the backend Data Turbine API (Java) with a front-end charting API (Javascript). This led to the research and use of JQuery/AJAX and Tomcat in conjunction with a Java servlet. Next it was necessary for us to find a front-end charting API. The first decision was to use Google Charts, but we eventually migrated when some of our fellow peers’ suggested the use of HighCharts. In order to view the charts, we needed to learn how to organize and design a website. After some research Twitter Bootstrap API was decided on. Finally, we needed to find out how to host the web application publicly in a cloud server so that it could be accessed remotely from multiple machines.

Requirements
The following lists comprise the set of marketing requirements necessary to our project:

Hardware
1. The system should recreate the given prototype as a printed circuit board (PCB) enabling a quicker, less error-prone building process.

Software
1. Easy to use and learn
2. Provide an aesthetically pleasing data-viewing mechanism
3. Displays channel (sensor) data should be displayed via a chart
4. Available remotely to those with privileges
5. Able to specify the Sensor Pod deployment to connect to
6. Able to specify the channel (sensor) from which to view data
7. Able to input the date and time from which to view data
8. Able to input the duration’s worth of data to view
9. Capable of real-time data updates to a sensor’s chart
10. Able to provide the user with the most recent available data for a particular channel
11. Uses open-source tools so that others could modify and add functionality
Specifications

Hardware
The system should implement the schematic displayed in Figure 4: UCSD Team's Schematic. It should be equal in size, or smaller than that of the prototype. It should take an input between 9-18VDC. At most, the system should use 11-12W of power. Regular operating should take 5-6W. It should sustain operation in an oceanic environment, temperatures between -10 and 45°C and humidity levels of 0-100% RH.

![Schematic Diagram]

**Figure 4: UCSD Team's Schematic**

Software
1. It should be hosted in a way that it can be publicly and remotely accessed
2. It must use the API provided by the Open Source Data Turbine Initiative for fetching data ([http://www.dataturbine.org/content/documentation](http://www.dataturbine.org/content/documentation))
3. It must only use open-source software APIs
4. It should connect to a valid provided host address and port number, i.e. 50.18.112.125 and 4444, respectively
5. It should be accessed from anywhere with an internet connect
6. It should be able to be viewed from Javascript enabled web-browsers
7. It should be able to select a duration as long as a week and short as the real-time update period
8. Charts should be capable of updates without user needing to manually refresh the webpage
Design

Hardware
The physical SensorPod is a PCB within a waterproof buoy, with sensors attached that hang in the water, as seen in Figure 1. An android phone reads sensor data using the onboard IOIO, and using a cellular network, sends that data back to UCSD's Open Source DataTurbine (OSDT) server.

It is a data-transmission unit in this above situation: the SensorPod engineers a route for a scientist to plug in appropriate sensors to the buoy and read the data while back at their computer.

There are three onboard sensors that monitor the SensorPod’s internal temperature, humidity, and voltage. Although the SensorPod design is independent of the external sensors, examples can be seen in the system block diagram in Figure 2: System Block Diagram.

At the beginning of the project, the UCSD team had a fully functional hardwired prototype developed. Our job was to recreate the given system in a CAD program, so that the wires could be replaced with PCB traces, permitting the board to clean up and condense.

Prototype (Winter Quarter, 2013)
The first quarter of the project was spent assembling the prototype. The DataTurbine group provided a schematic, shipped all necessary components, and a few pictures of one that they had assembled.

We opened and became familiar with the components, and likewise with the schematic wiring diagram. After speaking with our client, we tried to figure out how to place the parts on the prototype board. We learned about connecting the parts to the board. Our client sent us some pictures of his current setup. This manifested a vision of the prototype.

Austin was a teacher’s aid for Gary Perks' IME 156 course. The IME lab has everything necessary to design and fabricate a PCB. It also has plenty of soldering and crimping tools. During week 2 Gary clarified access privileges: we could use the lab if it’s unlocked, but must not use the lab alone, for safety in the case of an accident.

The career fair and follow-up interviews delayed hardware production. We screwed all of the components to the prototype board, crimped and connected the components as necessary, wired everything to the 15W DC/DC converter, and bought larger headers to connect the fuse-holder and to create hookups for the 12V input. We didn’t have a power supply, however the EE senior project provides a bench power supply. It also has Diptrace, the same CAD program that the IME lab computers have. That’s what we will use to design the PCB.
We tested the prototype using a multimeter, and finalized the product by adjusting any insecure or wrong connections. We got the Android source code from our client so that we could program the IOIO and the Android phone. Once we got the project loaded to a repository with BitBucket, we were able to

- Import projects to Eclipse
- Download necessary Android SDK's
- Update project settings, such as target SDK's
- Get android devices to link with Eclipse
- Install DataGather and DataLineProcessor4RemoteDT apps onto the android devices

These two applications use the IOIO Java API to gather the sensor data. They then transmit this data to UCSD’s Open Source Data Turbine [1] servers.

PCB (Spring Quarter, 2013)

*Using datasheets and calipers, I was able to build a Diptrace library of the SensorPod's components. These components were connected pertaining to the schematic given. The PWB was fabricated at an external source. I used through-pin headers to solder the components to the PWB, connected the power and the Android phone, and the project was complete.*
The first three weeks of the spring quarter included establishment with DipTrace. Resources included a Cal Poly-hosted tutorial [3]. Austin enrolled in a course that will walk me through a proper creation of the PCB.

A custom library of components was created within DipTrace. This involved precise measurements and took a bit of time. Each component was measured in millimeters using calipers. The edge dimensions were measured, as well as the location of the through-holes, so that a virtual version of the component could be created. This allowed a component with through-hole headers to be soldered onto the board beneath. With this library completed, plus the schematic in Figure 6. The final layout was generated, organized as preferred, and the traces were automatically routed.

![Diptrace PCB Layout](image)

**Figure 6: Diptrace PCB Layout**

The board was fabricated abroad and returned. Most of the components soldered on easily using through-hole headers. The on-board Phidgets sensors needed to have their current pins de-soldered. This was the toughest part of assembly and took a few tries. One component was destroyed in the learning process.
Connecting the phone to the board was complicated. The board and phone did not communicate well: they would reconnect indefinitely. This was fixed by adjusting the current via the on-board potentiometer.

Software
The task of the software aspect was to create a new web-based sink client to view the data from the sensors on the Sensor Pod. This first required the installation of all the necessary software such as the DataTurbine server and the local RDV application.

In order to get an idea for how a source, server, and sink communicated, we set up the DataTurbine server with a provided source and installed RDV to a local machine to view the mock data. The goal of the design was to mimic the capabilities of RDV, seen in Figure 8. These capabilities included the ability to select a date and time, a duration, a host address and port number, and a channel to chart.
After becoming more familiar with how the system communicated, we began to start thinking about how we could design the new sink client. This led to research about the common structure of web applications. Because the Data Turbine API is in Java and many of the charting APIs are Javascript-based, we were limited in our options. Originally the thought was to use Rhino [12] in an attempt to merge Java and Javascript into the same file. However, this proved to be overly complicated and not a viable option. We then spoke with a few of the sources our advisor, Professor Benson, provided (Doug Guastaferro and Jon Sehmer) to learn more about the front-end and back-end of web applications. We told them our situation and they recommended using Tomcat to host a back-end Java servlet. We then researched about using JQuery and AJAX with the front-end Javascript to request data (formatted in JSON) from the back-end Java servlet hosted on the Tomcat server.

The front-end of the web application is a collection of html files and an all-powerful Javascript file. The html files represent each page of the website, such as home, about, and documentation. They use the Twitter Bootstrap API to design the user interface and organize page elements. Examples from the webpage can be seen in Figure 9. Upon access to the home page, the Javascript file (draw.js) sits in the background handling user requests.
The Javascript file has many responsibilities and its flow diagram can be seen in Figure 10. Initially, draw.js obtains user input from any forms on the page and waits to be triggered by a click on a channel link. The input values and channel name are then sent via JQuery/AJAX post() to the DataFacilitator Servlet. It then waits for a 2D array representing a collection of data values and their respective timestamps. This represents the initial data series and once received, it is passed on to be organized and plotted. If no data is received an appropriate error notification is displayed. This charting is being done by using a dynamically updated chart from the HighStock charting API. If the user has selected the real-time updating option, the file sends another JQuery/AJAX post() to the DataFacilitator Servlet every two seconds for one unit of the most recent data. If there is no new data, the chart remains as is.
The “middle-end” design is contained in the DataFacilitator Java servlet, seen in Figure 11. The DataFacilitator is responsible for the communication between the front and back end of the web application. The majority of this functionality takes place in the doPost() method of the servlet. When it receives a data request from the front-end, it retrieves the JQuery/AJAX request parameters and uses them to create an instance of the back-end SensorDataRetriever class. It then calls for the back-end to return the requested data so it can be parsed to JSON and relayed back to the front-end.
for a specified time and duration. It’s called upon from the DataFacilitator servlet to retrieve sensor data from RBNB DataTurbine server. First a connection is opened to the server and a channel map is built with the given channel name. Its method goFetchData is called upon to begin the fetch process and return a SensorData object of two arrays filled with the requested time duration’s worth of data and its respective timestamps.

**Figure 12: Back-end Program Flow**
Testing/Verification

Hardware
The hardware was straightforward to test. After assembling the prototype and PCB, we tested each node using a multimeter to check that our components were taking in and sending out appropriate voltages per Table 1.

Table 1: Appropriate Component Voltages

<table>
<thead>
<tr>
<th>Component:</th>
<th>RS232 Shifter</th>
<th>DC/DC Converter</th>
<th>Voltage / Temp / Humidity Sensor</th>
<th>IOIO Power</th>
<th>IOIO Data lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Vin (VDC)</td>
<td>5</td>
<td>9-18</td>
<td>5V</td>
<td>5V</td>
<td>3.3V</td>
</tr>
<tr>
<td>Expected Vout (VDC)</td>
<td>5</td>
<td>±5</td>
<td>5V</td>
<td>USB power to phone</td>
<td>N/A</td>
</tr>
</tbody>
</table>

These values were gathered from datasheets that can be found in the bibliography under the “components” section.

After the voltages checked out, we connected the Android phone but there was a problem that could not be overcome. The Android phone would not switch itself into a USB-connected mode. It would try, over and over, and reconnect indefinitely. There was one place were we could have gone wrong: the schematic showed a capacitor across the input terminals of the DC-DC converter. We had access to various capacitors, and tried powering the system with each in place. Success came with the 47uF capacitor, but only once and never again.

When the Android phone finally connected to the IOIO, we launched the “Test IOIO Board” app that was provided by the UCSD team. This app is designed to display data from each of sensors. Our instance showed data from each of the on-board sensors, and as expected, zero data from the RS232’s, since no external sensors were plugged in. The internal sensors showed values of 22, 63, and 12, for temperature, humidity, and voltage, respectively. This confirmed that the on-board sensors were connected properly, and that the de-soldering of the factory-installed pins did not ruin the integrity of the components.

Note: the voltage appeared as a negative number. According to the user guide, “It is possible to connect the leads of the voltage source to either terminal - the reported voltage may just be of the opposite polarity” [8]. My fabricated PCB returns a negative polarity, so in the testing phase I updated the schematic and layout files accordingly.

Other testing-phase updates to the DipTrace CAD files include:
- Increased through-hole diameter for the headers
- Voltage sensor VIN+ and VIN- hole spacing was too close
- Place for capacitor in layout
- Added 10K and 20K resistor labels
- Added + and - labels to the header that takes in DC power
- Added mounting holes on all four corners

Final note: although my measurements of the IOIO through-holes were taken with precise calipers, my PCB did not line up perfectly. When assembling a new board, make sure to line up all headers and screw holes before soldering at all. I messed up because my initial set of headers was soldered on at a slight angle, therefore causing me to angle all future headers.

Software
Because keeping an open instance of the cloud server currently being used (Amazon Web Services) costs money, much of the testing was done using Eclipse, a local Tomcat instance, and various web developer tools such as Mozilla Firefox’s Firebug extension. The local Tomcat instance allowed us to open and view the web application in any browser on the computer (mainly Chrome and Firefox). While in the browser using Firebug, it was possible to detect server communication errors, debug invalid Javascript and HTML, and follow the application as it progressed through its states. The Eclipse console was also useful for debugging the Java servlet in order to verify that the back-end was getting the correct request parameters from the front-end. It was also used to find out why real-time updates weren’t acting as expected, as it led to learning that the “newest” piece of data didn’t necessarily mean the current time. It instead meant the last time that sensor took a data sample. It also led to learning the different periods that different channels sampled data at. This made it necessary to pay attention the returned timestamp of the points as well, instead of deciding our own time duration between points to display.

Another verification method once the site was complete was to use a known host address and port number of a currently deployed system. The main one used is stationed at the Scripps pier down by UCSD. We opened both the old RDV program on our computer and the new web application, connecting both to the above deployment and inputting the same Date/Time and duration. The output of the graphs displayed the same data points (See Figures 13 & 14). The reason for the apparent discrepancies in these figures is due to the fact that the scales are different, as the y-axis Figure 14 is in the range from 22.45 to 23.6. The y-axis for Figure 13 ranges from 0 to 23, making the grade of the lines to appear different. A new functionality is also shown in Figure 13 that the web application gives in comparison to its predecessor. This is the ability to hover over the various points of the graph to have it display the data value and time for that point.

If the real-time option is set for the internal sensors of the web application, the chart should update accordingly every minute with a new reasonable data value that fit in the range of the past measurements.
Figure 13: Eco Sensor Pod Web App Output
Conclusion

Hardware
This project began with a schematic and ended with a clean, deployable PCB. The project walked us through a process that was very slow the first time through. Now that we’ve had experience designing in Diptrace, we have that on our toolbelts forever. It’s a new tool that we can apply quickly and easily. With the CAD experience came some strenuous de-soldering experience, which is a skill that should always be appreciated. Finally, there’s nothing more satisfying than powering up a PCB after learning about each and every step that went into its production.

In a future version, some components should be changed: use a (PCB mount) DC power jack and use on-board sensors that do not require de-soldering. For a future version that includes primarily surface mount (SMT) soldering, use the open-source schematics to design the components directly into the PWB, rather than using the current screw + through-hole method.

Software
One of the greatest experiences from the project was learning the beginning to end development of a web application. It took the knowledge of past work with Java combined with new languages and tools such as JQuery, Javascript, and Tomcat to put it all together. It was
great to look at the old system to see what could be improved or reproduced. Future work could include making the web application more robust and able to inform the user of its specific onboard sensor channels. It could also use a bit of refactoring on the front-end side for submitting form data and triggering chart updates. It could also be extended to overlay similar series data on the same chart. There could be a world map option to show the currently deployed sensor pods so you don’t have to manually input the host address and port number.
Bibliography

1. DipTrace Tutorial
   http://idesign.calpoly.edu/dip-trace/
2. RoHS
   http://www.rohscompliancedefinition.com
3. EcoSMES.net
   http://www.ecosmes.net/cm/navContents?l=EN&navID=eee&subNavID=2&pagID=18&flag=1
4. DataTurbine
   http://www.dataturbine.org

Hardware Components

5. DC/DC Converter
6. IOIO
   https://www.sparkfun.com/products/10585
7. RS232 Shifter
   https://www.sparkfun.com/products/449
8. Voltage Sensor
9. Humidity / Temperature Sensor

Software Research

10. DataTurbine API
11. Web Application Design
    http://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend
12. Scripting with Rhino
13. Converting Java objects to JSON
    https://code.google.com/p/google-gson/
14. AJAX Fundamentals
15. Deploying Projects as WAR files with Eclipse
16. Tomcat Basics
    http://oreilly.com/java/archive/tomcat-tips.html
17. Overview on how Servlets Function
   http://shivasoft.in/blog/java/servlet/how-container-handles-the-servlet-request/
18. API for Charting
   http://api.highcharts.com/highstock#plotOptions.series
19. More Discussion about Servlets
   http://www.doc.ic.ac.uk/csg-old/java/servlets/servlets.html
20. The Post() Method of JQuery
   http://api.jquery.com/jQuery.post/
21. Webpage Design Tutorial
   http://www.w3resource.com/twitter-bootstrap/tutorial.php
22. RDV Spec
   https://nees.org/tools/rdv/wiki/KeyFeatures
23. Helpful Notes on DataTurbine API (Especially for sink requests)
24. Cloud server the project is being hosted on
   http://aws.amazon.com/ec2/
25. How to set up the AWS server
26. More AWS setup
   http://developer24hours.blogspot.ca/2012/12/install-tomcat-7-on-amazon-ec2-ubuntu.html
Appendices

A - Senior Project Analysis

Analysis of Senior Project Design

Project Title: Eco Sensor Pod Quarter / Year Submitted: Spring 2013

Student: (Print Name) Michael Chamoures (Sign) ______________________

Student: (Print Name) Austin Zuffi (Sign) ______________________

Advisor: (Print Name) Bridget Benson (Initial) _______ Date: ________

Summary of Functional Requirements

Hardware
1. The system should recreate the given prototype as a printed circuit board (PCB) enabling a quicker, less error-prone building process.

Software
1. Easy to use and learn
2. Provide an aesthetically pleasing data-viewing mechanism
3. Channel (sensor) data should be displayed via a chart
4. Available remotely to those with privileges
5. Able to specify the Sensor Pod deployment to connect to
6. Able to specify the channel (sensor) from which to view data
7. Able to input the date and time from which to view data
8. Able to input the duration’s worth of data to view
9. Capable of real-time data updates to a sensor’s chart
10. Able to provide the user with the most recent available data for a particular channel
11. Use open-source tools so that others could modify and add functionality

Primary Constraints
Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?
The toughest part about building the hardware was desoldering the Phidgets on-board sensors. Those factory-soldered pins did not want to slip free, and we broke one voltage sensor in the process.

**Economic**

- Original estimated cost of component parts (as of the start of your project)
  - Unknown
- Actual final cost of component parts (at the end of your project)
  - Unknown
- **Attach a final bill of materials for all components**
  - See Appendix B
- Additional equipment costs
  - PWB fabrication taken care of as part of IME 458: price unknown. If ordered from within DipTrace, each unit ordered costs $65.49.
  - Solder and solder wick attained from the EE and IME labs ~$3
  - Crimps ~$10
  - Power supply for testing
  - Diptrace ~Free
- Original estimated development time (as of the start of your project)
  - 6 hours to build the prototype board
  - 40 hours to produce PCB
  - 100 hours on software
- Actual development time (at the end of your project)
  - 30 hours to build prototype board
  - 60 hours to produce PCB
  - 150 hours on software

**Environmental**

The physical product uses RoHS compliant components and consumes no more than 12W of power.

The use of this project will provide ecologists and other scientists the ability to monitor environmental sensor readings from a remote location in real-time. What they do with this information is at their discretion.

**Manufacturability**

Manufacturing the PWB was performed by an external source for an unknown price. There should not be any other challenges: simply export the Gerber and NC/Drill files from the Diptrace PCB Layout and send them to a manufacturer. You can also order from within Diptrace, File > Order PCB, from BayAreaCircuits for a price of $65.49 per unit.

**Sustainability**
Hardware
This product should sustain until overheated or destroyed due to water damage. The product should thrive at a reasonable power consumption <= 12W. Components are RoHS compliant and therefore are made without lead or other dangerous substances [2]. Design upgrades, as mentioned in the conclusion, are purely practical and do not influence the sustainability of the product. Upgrades might include swapping out the on-board sensors, adopting a standard power connection, or implementing surface mount technology (SMT). If these upgrades were to be made, another board would have to be produced, making a significant environmental impact, as described at ecosmes.net [3].

Software
In order to keep the project running, it must be hosted on some sort of server. If the goal is to always have the site public then there must be some sort of cloud instance holding it. The software side of the project was designed using only open-source technologies so that it could be free, extended and modified in the future. Some of this future work could include making the web application more robust and able to inform the user of its specific onboard sensor channels. It could also use a bit of refactoring on the front-end side for submitting form data and triggering chart updates. It could also be extended to overlay similar series data on the same chart. There could be a world map option to show the currently deployed sensor pods so you don’t have to manually input the host address and port number.

Ethical
These devices are going to gather information about the health of oceans and lakes. It can be argued that the public should be aware of this information. This information should not be abused in return for profit or for any interest that involves hiding poor health of the water.

Software
Because the project is hosted publicly, someone unauthorized could view the data being collected. If onboard sensors are instead used to monitor something illegal, this could also be a problem as the web application would be displaying that data.

Health and Safety

Hardware
If produced using non-RoHS compliant materials, as electronic waste, these minerals will seep through the earth into the groundwater and might return in the local tap water.

Software
There are not really any health and safety concerns with the web application.

Social and Political
Similar to ethical concerns, information about public water should not be abused in return for profit or for any interest that involves hiding poor health of the water.

Development
Describe any new tools or techniques used for either development or analysis that you learned independently during the course of your project.

**Hardware**
- Diptrace for computer aided design of a printed circuit board

**Software**
- Java servlets / web applications
- Javascript / JQuery / AJAX / JSON
- Tomcat7
- Amazon Web Services
- HighCharts interactive javascript charts
- Twitter Bootstrap
  - It’s a free collection of tools for creating websites and web applications. It contains HTML and CSS-based design templates
- Mozilla Firefox Firebug plugin
  Used for debugging web app in the browser
### B - Parts list

<table>
<thead>
<tr>
<th>Received From</th>
<th>Qty Req’d</th>
<th>Description</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>JameCo</td>
<td>9</td>
<td>Jack Screws, Nuts, Washers Set</td>
<td>For a total of 18 screw connections</td>
</tr>
<tr>
<td>JameCo</td>
<td>1</td>
<td>Pwr Supply, DC-DC, Encap, 15W</td>
<td><a href="http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_1001_155708_-1">http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_1001_155708_-1</a></td>
</tr>
<tr>
<td>BlutekUSA</td>
<td>1</td>
<td>Motorola MicroUSB cable with travel charger</td>
<td></td>
</tr>
<tr>
<td>Phidgets Inc.</td>
<td>1</td>
<td>Humidity/Temperature Sensors Item 1125_0</td>
<td></td>
</tr>
<tr>
<td>Phidgets Inc.</td>
<td>1</td>
<td>Precision Voltage Sensors Item 1135_0</td>
<td><a href="http://www.phidgets.com/products.php?product_id=1135_0">http://www.phidgets.com/products.php?product_id=1135_0</a></td>
</tr>
<tr>
<td>Amazon</td>
<td>1</td>
<td>Littelfuse MIN3 Fuse (5-pack)</td>
<td></td>
</tr>
<tr>
<td>Amazon</td>
<td>1</td>
<td>Littelfuse In-Line Fuse Holder</td>
<td></td>
</tr>
<tr>
<td>UCSD DataTurbine</td>
<td>1</td>
<td>Android Device</td>
<td></td>
</tr>
<tr>
<td>SparkFun</td>
<td>1</td>
<td>IOIO for Android</td>
<td></td>
</tr>
<tr>
<td>SparkFun</td>
<td>3</td>
<td>9 Pin Female Serial Connector - PCB Mount</td>
<td></td>
</tr>
<tr>
<td>SparkFun</td>
<td>3</td>
<td>RS232 Shifter SMD No DB9</td>
<td><a href="https://www.sparkfun.com/products/449">https://www.sparkfun.com/products/449</a></td>
</tr>
<tr>
<td>SparkFun</td>
<td>2</td>
<td>Screw - Philips Head (1/4&quot;, 4-40, 10 pack)</td>
<td>For a total of 18 screw connections</td>
</tr>
<tr>
<td>SparkFun</td>
<td>2</td>
<td>Screw Terminals 3.5mm Pitch (2-Pin)</td>
<td><a href="https://www.sparkfun.com/products/8084">https://www.sparkfun.com/products/8084</a></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>10K•Ohm resistor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>20K•Ohm resistor</td>
<td></td>
</tr>
</tbody>
</table>
C - Project code

The full repository for the code is hosted on both a public BitBucket account and a public GitHub account. The addresses for these are as follows, respectively:
https://bitbucket.org/ecosensorpod/eco-sensorpod.servlet/src
https://github.com/CalPolySensorPod/Eco-Sensor-Pod-Servlet.git
D- User Manual

Hardware

Edit CAD files, if appropriate, using Diptrace, downloadable for free at their website.

If you edit the schematic, convert to PCB using ctrl+B, organize the components as you desire, and use autoroute. It will automatically re-check the design rules and all should be well. After the design check, the via sizes used to throw error. I adjusted the values for the default via, and the DRC now checks out perfectly after an autoroute.

Export Gerber and NC/Drill files. Send those to a manufacturer and have your finished board shipped to you. Else, fabricate it yourself.

This design requires through-hole headers to solder on the components. The toughest part of the assembly process is preparing the on-board sensors for through-hole soldering. The Phidgets sensors come with a wired VCC, GND, and data line. My design requires that these be de-soldered and replaced with through-hole headers. The desoldering on these particular boards was significantly more challenging that de-soldering that I’ve done in the past: either the pins are too large for the through-holes, or the solder has a higher melting temperature than usual. Once accomplished, solder onto the proper location.

One of the headers takes a direct VCC and GND, indicated by a “+” and “-” on the layout. The other header takes the fuse line.

Two holes exist in the layout titled as “OPT” for optional. This is for a capacitor. On the schematic in Figure 4, there is a capacitor between the positive and negative input terminals. If you know what capacitor value to add this circuit, add it there for stability, else, leave it out like we did.

Software

To use the web application:

- Start by testing that your computer (or cell phone, tablet, etc.) has an internet connection. Next, using a web browser of your choice (i.e. Google Chrome, Mozilla Firefox, Safari), access the web application by entering the following address into the URL bar:
  http://54.215.6.156/EcoSensorPodServlet/home

  Once it loads you should see the homepage, shown in Figure 15:
From here you can learn more about the project or choose to connect to a sensor pod to view data.

Clicking the “Learn More” button will take you to the about page (Figure 16) where it explains some of the different tools used in both the hardware and software that power the system:

If you decide to view data you must do the following:

- Enter a valid host address and port number of the desired sensor pod in the forms at the top left. These should be in the forms of x.x.x.x and ####, respectively, as seen in Figure 17:
Next, choose a duration’s worth of data to plot from the “duration” dropdown located at the top right of the page, as seen in Figure 18.

The next step depends on how you have real-time updates configured at the top center of the home page, as seen in Figure 19.

- If “Yes” is selected, then it will plot the chosen duration’s worth of data preceding the current time. Skip ahead to next step
- If “No” is selected, then you must specify an end date and time located just to the right of the Host Address form, as seen in Figure 20.

The plotting settings will be set upon clicking on a channel name link. These are located midway down the left side of the webpage, as seen in Figure 21.
Figure 21: Example of Channel Name List Links

- Once the link is clicked, a chart should appear just to the right of the channel links if all information was inputted correctly, e.g. Figure 22

![Chart Image]

Figure 22: Example Output from Web App

- If a chart does not appear, the web page should warn you with a pop-up box informing the problem
- Important errors include:
  - There is no data on the desired channel during the specified time period. If you get this, it should inform you of the end time of the most recent available data, if any, for that channel on the RBNB server
• Use the zoom selector at the bottom of the chart to magnify certain areas of the chart, or use the buttons at the top left of the chart to automatically resize the displayed period
• If you’d like to save the current image of a specific chart, you have the option at the top right of the chart to save it as various types of extensions, or print it
• Be aware that if real-time updates are selected, although the web application may request new data every couple of seconds, there may not necessarily have been new data sampled in that period. If this is the case then the chart will appear unchanged
• Additionally, if the real-time update option is set you have the option to pause and play them with the check-boxes just above the chart, as seen Figure 23

Figure 23: Real-Time Update Control Options