iOcean RGB Sensor

Mark McNeff and Archit Mendiratta
Advisor: Bridget Benson, Electrical Engineering
Cal Poly State University San Luis Obispo
Electrical Engineering Program
April 24th 2015

Abstract

The goal of this project is to create an accurate ocean color sensor that will communicate effectively with an iPhone. The device will take ocean color measurements at its current location and provide these measurements to the user for scientific study of the variability of ocean color across the globe. The data gathered by the device will also provide ground-truthing of satellite ocean color measurements.

Acknowledgements

We would like to thank our project advisor, Dr. Bridget Benson, for her sincere help and supervision of our project design and implementation.

We would like to thank Dr. Thomas Bensky for his knowledge and guidance with the optical properties of this project and giving us good insight into potential designs as well as giving us access to optical lab equipment.

We would also like to thank Dr. Dennis Derickson, the chair of the Electrical Engineering department for his help in the early stages of this project.

Finally we would like to thank Dr. Brian Lottman and Northrop Grumman Corporation for sponsoring this project and partnering with Cal Poly in this project.

Table of Contents

List of Tables and Figures

Tables:

Figures:

Introduction

Due to overfishing, pollution, and climate change, ocean health has been deteriorating. About seven months ago, scientists assigned a score of 67/100 to the Earth's ocean health by evaluating each region's ecological, social, economic, and political factors (Howard).

Recently, countries have started making more informed decisions concerning the ocean's health and the score has been slowly on the rise. Microscopic plants, for example, are the primary source of nutrients for the ocean ecosystem and thus, are important in determining ocean health. Many microscopic plants, such as phytoplankton, alter the ocean's color (San Diego State University). Therefore, monitoring ocean color plays an important role in determining the flux in the ocean health. Accurate readings allow for better comparisons among different ocean regions, or among different time periods of the same region.

Other factors that contribute to ocean color include both artificial and organic products, and, although minimally, ocean depth. These products can be factory waste, oil, plastic, dirt, dust, animal waste/decay, or other materials. These particles and materials become suspended in the water and can affect ocean color readings if a sensor is reading from the same location.

Future NASA PACE and ACE missions will collect ocean color and atmospheric information from low earth orbit to understand ocean health.[5] The aerospace industry, scientific, and academic communities will partner to develop the space, ground, and supporting elements that produce the best science. As of now ocean color information as observed by satellites represents a very small fraction of the received signal, and is difficult to extract accurately. The space-based measurements must be calibrated and validated by instruments on the ground.

Future systems will begin to rely more on citizen scientists using low cost portable, ubiquitous technology. This is the major group that this product would be marketable to. Although this project is specifically for NGC, it could be more broadly marketed to the scientific community at large.

Right now there are many satellites monitoring the oceans and taking a plethora of data about them, as shown in Figure 1. However, several measurements are very hard to accurately extract at the moment. There is a lot of data manipulation that needs to occur in order to extract things like the color at a specific spot. This is partially due to the atmospheric distortion that happens by the time the data gets up to a satellite. Also, the coastlines provide particular difficulty due to the fact that one pixel on a satellite may contain miles of both land and water. This stark contrast makes it difficult to obtain an accurate color and atmospheric reading over just the ocean part of the pixel.

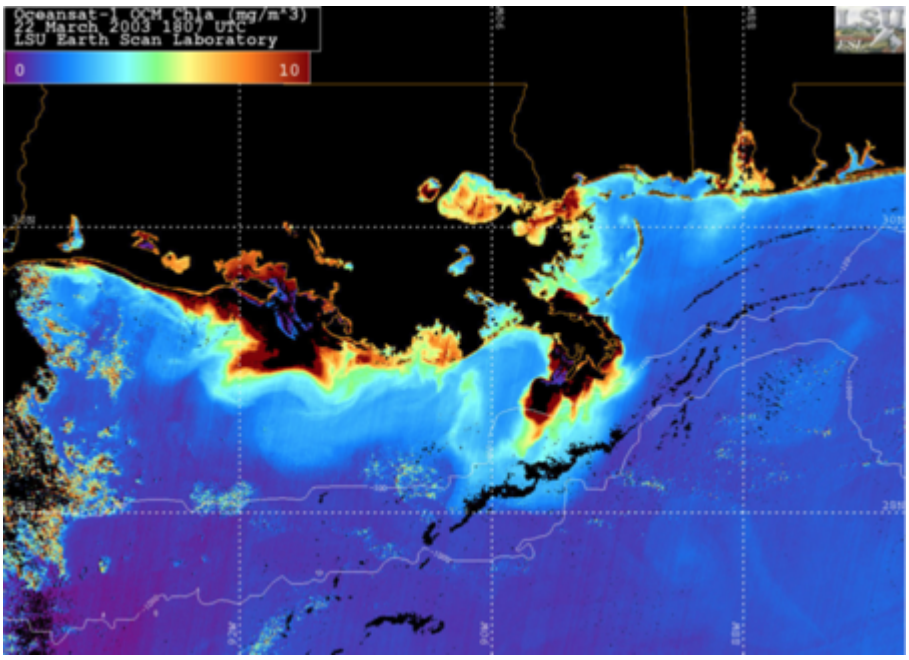

Figure 1: Ocean Satellite Color Image

As of now the market really does not have a product like this. The SIMBAD radiometer is a piece of similar technology, however since it was released in 2004 both electronics and sensors have improved drastically. There is also the Hydrocolor app, which utilizes the camera's phone instead of an external sensor. If and when further specifications for this project are received from NGC, we will be able to better market this to a specific people group. However the small, independent, research groups are market that is seen as an easy market to tap into and to advertise to. Of course with the development of smart phones there will be a significant difference in the designs. The fact that there is really not any competitive products available gives our team, and by extension Northrop, a very big advantage when designing a satellite or contract with companies and organizations, such as NASA, on the ocean data front. The satellites that are currently in use that are specifically for oceanography are the SeaWiFS (Sea Wide Field of view Sensor) and the POLDER (**POL**arization and **D**irectionality of the **E**arth's **R**eflectances) project. An image from SeaWiFS is shown below

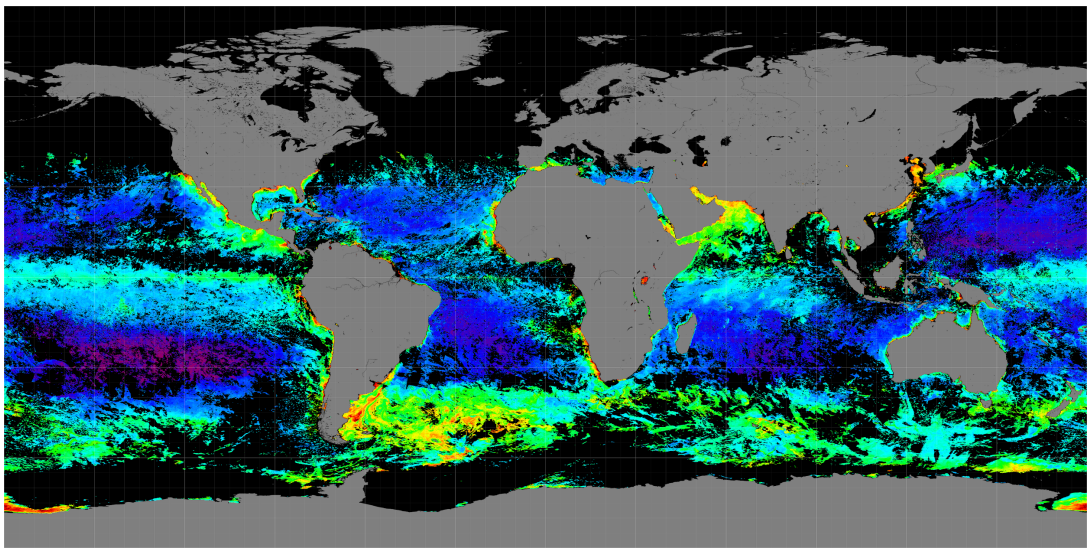

Figure 2: SeaWiFS December 2010 Image

Our product will use the already well-documented IOP's of seawater, take coastal readings of aerosol thickness, and measure a digital RGB value for the surface of the water. It will then send this RGB value to a microprocessor for analyzing. Finally, the analyzed data will be sent via Bluetooth LE to the connected iPhone and displayed in an intuitively programmed app that will allow the use to make inferences and reports on the condition of the point of measurement. Figure 2 shows another satellite image that measures the ocean RGB values.

Requirements

Given the time, resources, and other constraints we had when designing and implementing the system, these are the requirements that were stated by NGC and were deemed most important in creating "iOcean."

- Contains to main components:
  - Portable hardware that records the data from the reflected ocean water light
  - Mobile application that displays and exports the recorded results from the hardware
- Provides accurate measurements of the following properties of the ocean's surface:
  - RGB - wavelength
  - Lux - Intensity
  - Time, Date, GPS coordinates

- Uses off the shelf components making a scalable design
- Communicates with iOS device via a Bluetooth LE module
- Is relatively cheap to appeal to the desired market of independent or poorly funded scientists

In order for this to be a viable and practical product it shall be small enough to be held while attached to a smart phone, and shall be able to transmit data to the smart phone as well. It shall receive measurements via the sensing apparatus and obtain a digital RGB value, which will correspond to a specific spectral wavelength of light as well as a lux reading to correspond to intensity. Taking, analyzing, and storing the measurements is the crucial functionality of this project.

Background

There are several themes and main components to this project that need to be addressed before we continue to the details of the design and report. There are the certain properties of light that are mentioned frequently throughout this report. These concepts are all quite complex but we will discuss them to a necessary degree here. The measurements we have mentioned above need to be defined and quantified so as to have full comprehension of what we desire to measure. In particular, the measurements that are of most importance and the ones that will need the most defining are RGB values and lux. These terms are somewhat vague and can mean different things to different people depending on context. So for the purposes of this project and report we will define these measurements as follows.

First it is worth talking about the properties of light and its energies. Light is understood to act as a wave, or for a 2 dimensional view a ray, of energy. It is also a particle simultaneously but that would be for a different report. This energy level, will correspond to a particular frequency in a positive correlation. More energy means a higher frequency. This frequency corresponds to a wavelength with an inverse correlation. The higher the frequency the lower the wavelength. This wavelength will put the light on what is called the light spectrum. It ranges from cosmic rays of extremely high energy and low wavelength to radio waves of very low energy and large wavelength. However the part of the spectrum we are most interested in is what is called the visible spectrum. The spectrum, with a breakout for the visible colors is shown below in Figure 3.

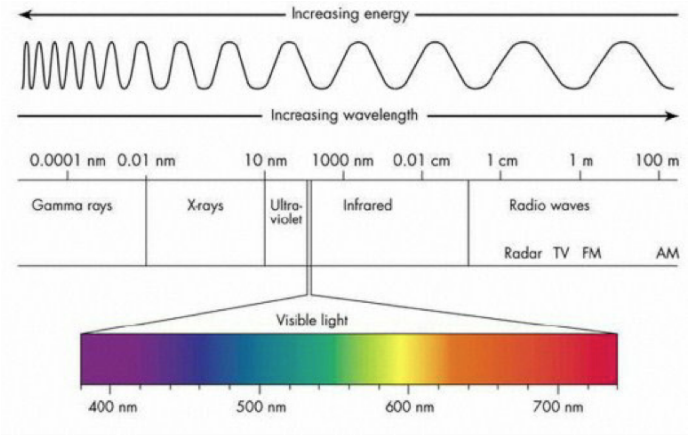

Figure 3: Light Spectrum

This is the part of the spectrum that gives every object and thing we see its color. The color something appears to be is the color that is reflected by it. Or conversely you could say that something that looks blue absorbs all colors except blue. So, when we are looking at ocean color what we mean is the following. Light from the sun of various different energies strikes the surface of the ocean, many of these light energies are absorbed, however some are reflected. These reflected energies correspond to certain wavelengths that we see as different colors. There are also non-visible energies reflected and absorbed however these are not important to our scope.

In order to measure these energies, and thus color, we must take this light and put it into terms that we can correlate to what we see. This is what the TCS34725 sensor does. The sensor utilizes the ability of an LED to not only emit a specific visible light but also sense light of that same color. When connected to a circuit in the correct way an LED can act as a photodiode.

The LED array within the sensor is made of red, green and blue diodes and is hit by light that has been reflected off of the surface of the ocean's surface. This light is corresponds to the energies that were reflected off of the surface of the ocean and as such is "carrying" the properties of the ocean at a particular location. When the sensor reads this light the diodes will turn on and create a voltage proportional to the amount of light that hit it. This voltage is then transferred via an analog to digital converter to become what we call the RGB values.

The measurement of lux is proportional to the amount of total energy hitting the sensor. While the red, green, and blue diodes sense particular wavelengths the clear, or white, diodes on the sensor are a total of all visible wavelengths.

Lux is defined as the SI (Système Internationale or International System) unit of illuminance, equal to one lumen per square meter. A lumen is defined as the SI unit of luminous flux, equal to the amount of light emitted per second in a unit solid angle of one

steradian, or a 3D radian, from a uniform source (both definitions found via Google search definition). The sun is considered a uniform source and so the sensor takes the total amount of light sensed per second at a solid angle and extrapolates to give a value for the lux. However the sensor has a programmable gain which allows a longer amount of collecting to happen and will make the values seem like there is more light. So for any given application the sensor will need to be programmed against a standard measurement from another sensor.

The sensing of color is an art in and of itself; there are many different scales, standards, and schemes that have been used to classify, distinguish and determine color. It is as much art as it is science. When making inferences and analyzing the data we collect we will be using what is visually apparent and can be distinguished by human eye. This decision was made based on how and where the sensor will be used. It is a handheld device that is used for ocean observance by humans and thus we will use our eyes as what is “true” color.

Design

This section describes our hardware and software choices for the iOcean sensor and app. We went through 3 separate and unique design iterations however the final iteration is the design described in detail below. The hardware works together with the software to obtain data and communicate with the iPhone as well as send commands from the iPhone to the sensor. We combined knowledge of embedded systems, iOS programming, and physics in order to obtain the correct data and process it in an efficient manner.

Hardware

This section describes the hardware components, hardware considerations and circuit diagram of the iOcean Sensor. Table 1 lists each component used to make up the hardware portion.

Hardware Components

Quantity	Name
1	TCS34725 RGB Sensor Breakout board
7	Headers
1	HM-10 BLE Module
1	Arduino Uno with AtMega328 chip
1	iPhone 5
1	Battery Pack
1	Protoboard
1 ft.	Solder

Table 1. Hardware Components

Hardware Considerations

When the requirements were considered we were guided into choosing specific hardware components. These choices are reflected and described in detail below.

AtMega328 Microprocessor

We determined the AtMega microprocessor on the Arduino Uno board was an appropriate processor to use for the iOcean because of its

- Pricing
    - One of the requirements is that the price of the completed sensor is relatively cheap as its target audience is those without much if any funding. The AtMega on an Arduino board is very inexpensive and readily available for purchase
- Ease/Documentation
    - There is a lot of open source code available to get started with arduino based coding, including libraries and tutorials on performing specific functions.
- Analog and Digital I/O Capabilities
    - The fact that the chip and use the same analog or digital pin as an input or output within the same program is very useful. Also the ability to change the direction (input vs. output) of a pin mid program was very desirable as well.
- Bluetooth Functionality
    - The AtMega has the ability, especially when on an arduino board, to communicate well with Bluetooth devices. This is crucial for the completion of our project.

TCS34725 RGB Sensor

We decided on the TCS34725 RGB sensor for several reasons. Our first consideration was using an array of 5mm LED's however these were too bulky and would require additional circuitry. We then thought of using the iPhone's camera, but several factors persuaded us otherwise. There is a "purple haze" of sorts when trying to resolve very close bands of light as blue light is transmitted through the glass on the iPhone much more than green or red. Secondly, there is also potential to obtain infrared wavelengths in the reading, which will skew pure color as well. We then found this sensor, with an IR filter and an array of LED's that pick up red, green, blue, and clear (white or all colors) light using the clear to have a baseline of luminance when trying to isolate a color. Figure 4 shows the black box diagram of the TCS sensor. The actual image of the sensor is seen in Figure 5.

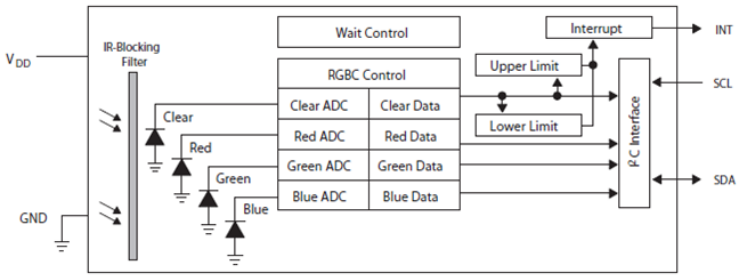

Figure 4: TCS Sensor Black Box Diagram

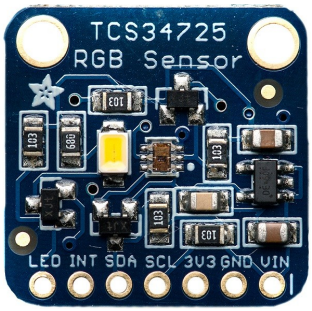

Figure 5: TCS Sensor with Surrounding Circuitry

Black Widow Long Range BLE Module

Only Bluetooth Low Energy Devices are supported on iOS devices without jail-breaking or otherwise rooting the phone.

iPhone 5

We selected an iPhone because in order to have native Bluetooth options for iOS devices we needed a 4s or newer and we already owned an iPhone 5.

Total System

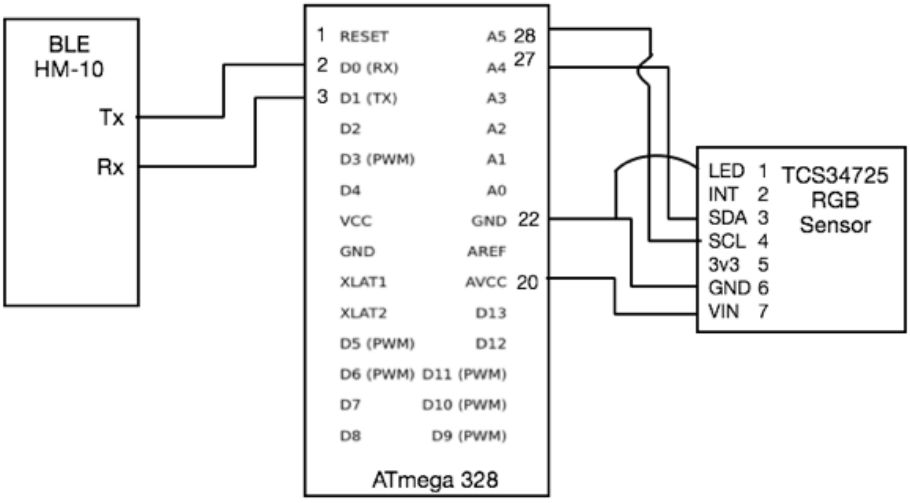

Figure 6: Full Hardware System

Software

All of the software aspect of this project was written in C and Swift. The code pertaining to the Arduino and BLE was written in C in the Arduino Software development environment. The iOS code was written and designed completely in Swift in the Apple Xcode environment. This section describes the software flow and specifications.

µ-Controller Software Flow

The AtMega/Arduino programming flow is actually quite simple. There are three major components to the software's organization; these components are the initialization, the Bluetooth link, and data collection/transmission.

The first stage is the initialization stage. This will be where the sensor parameters for collecting data are set, analog and digital ports are designated as inputs and outputs, libraries will be included, and definitions will be established. This step is crucial because if a port in not set up correctly or the sensor is put into a wrong mode, for the duration of the code there will be errors. It is vital to know exactly what we want the hardware to do so that we set up everything correctly in order to accomplish it.

The second stage is the Bluetooth link. This stage will test whether or not the Bluetooth module is linked to the iOS device. Without being linked to the phone there will be nowhere for the data to be sent or stored. This will also validate that the sensor is indeed taking data as it is supposed to.

Lastly is the data collection and transmission. Luckily the sensor has a built in library of functions that allow the user to easily collect RGBC color data, lux readings, and the color temperature. With this data the microprocessor can then analyze it to find other useful pieces of data such as dominant hue, wavelengths, and intensity. This data is then sent via the Bluetooth link to the phone so that the data can be collated and displayed in an intuitive manner in the custom app. The full software full diagram for the Arduino is shown in Figure 7.

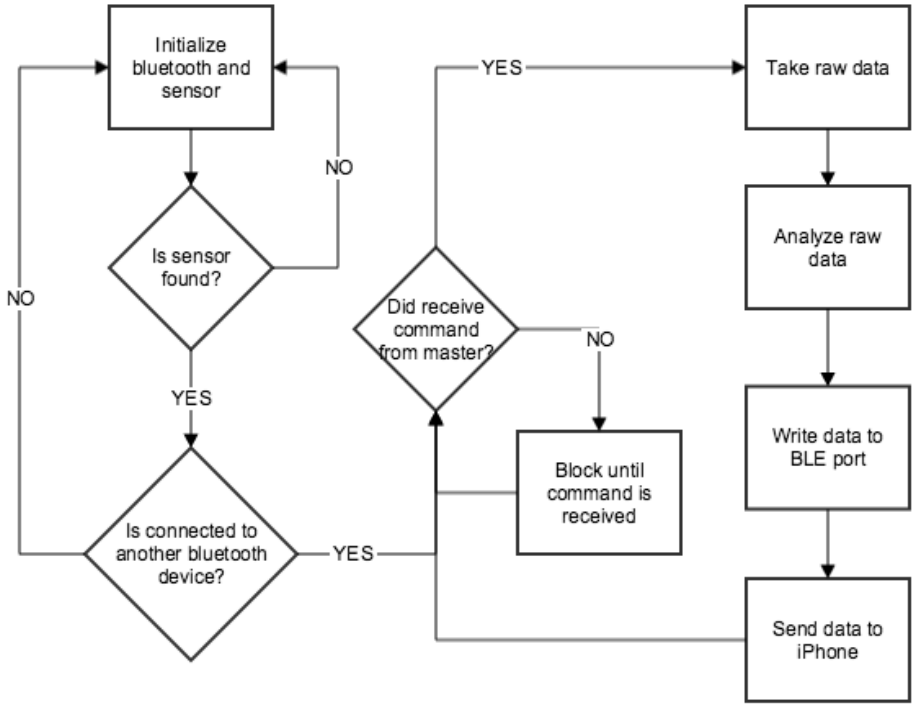

Figure 7. Arduino Software Flow Diagram

iOS Software Flow

The iOcean RGB sensor requires two hardware components, the Arduino with BLE shield and an iPhone 5 (or better). These two components communicate with each other through Bluetooth, the Arduino's BLE shield and the iPhone's built-in Bluetooth. Figure 8 shows the software flow diagram of the iOS application.

A custom-built iPhone application acts as the master device while the Arduino is the slave device. The app will automatically connect to the hardware when the app opens. The app has a "Collect" button that, when pressed, will collect the data from the hardware and takes the user to the data name screen so they can name the data collect. If the app is disconnected from the hardware, a message will appear to let the user know that they cannot collect data because the hardware is not found. The user can then enter a name for the capture recorded from the Arduino and can press "Done" to save the capture

or press “Cancel” to delete the capture. The “Cancel” button will take the user back to the capture table, but the “Done” button will take the user directly to the newly captured results.

There is another button called “View Results” that displays all of the previously recorded data. The data is persistent so even after the app is closed, all of the user’s data will be saved for when the user needs it again. The data is displayed as a single-column table where each row represents one capture from the Arduino. The user can interact with the data in multiple ways. Each row can be swiped left for easy deletion. When a row is tapped, a new window will appear that gives all of the details about the specific capture, including but not limited to the date and time or recording, RGB values, the lux, the color temperature, and the gps coordinates of the capture. The “Share” button, located at the bottom of the capture data table, allows the user to email all of the results to anyone, including the user himself/herself.

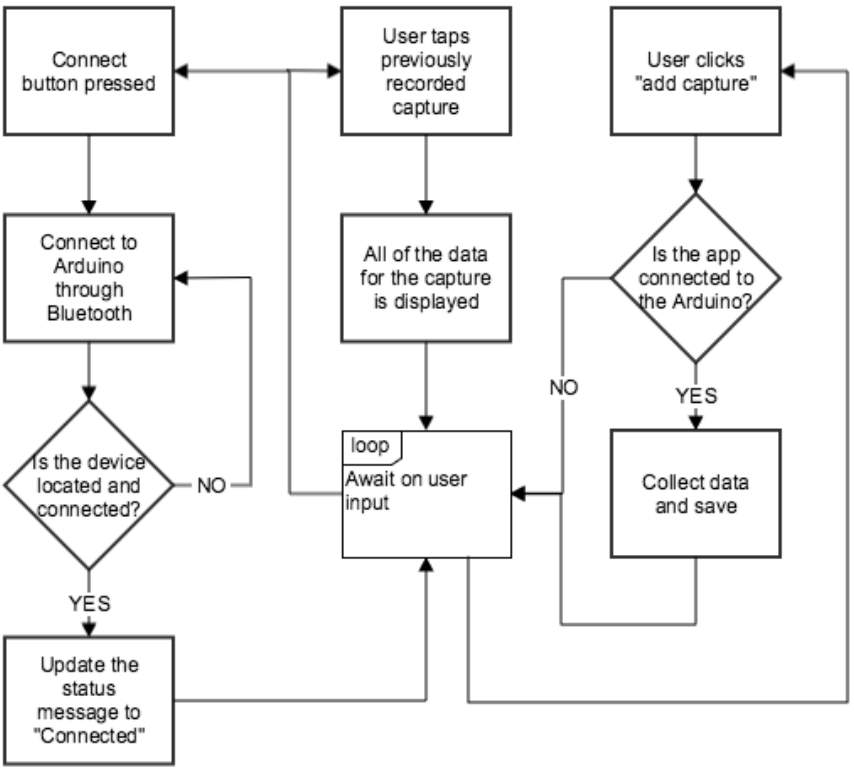

Figure 8. iOS Swift Software Flow Diagram

Testing

We were able to test our sensor under various conditions to ensure that the photo diode array was working properly. These results are uncalibrated but do well to show the range and consistency of the sensor.

Sensor Tests

We tested our sensor under 3 different lighting conditions a fully lit room, a halfway-lit room and a fully dark room. we know the room was exactly half lit do to the construction of the light switches. With one switch on and one switch off the room went to half power and with both switches off the room went fully dark. These test, while they are not from reflected sunlight off the surface of the ocean, allow us to predict what the results will entail. The results are shown below in figures 9 to 12.

Test Case #1: Fully Lit Room

```
Found sensor
Color Temp: 4095 K - Lux: 841 - R: 878 G: 1002 B: 622 C: 2694
Color Temp: 4092 K - Lux: 840 - R: 879 G: 1002 B: 622 C: 2695
Color Temp: 4095 K - Lux: 841 - R: 878 G: 1002 B: 622 C: 2694
Color Temp: 4095 K - Lux: 841 - R: 878 G: 1002 B: 622 C: 2694
Color Temp: 4095 K - Lux: 841 - R: 878 G: 1002 B: 622 C: 2694
Color Temp: 4092 K - Lux: 840 - R: 879 G: 1002 B: 622 C: 2695
Color Temp: 4092 K - Lux: 840 - R: 879 G: 1002 B: 622 C: 2695
Color Temp: 4095 K - Lux: 841 - R: 878 G: 1002 B: 622 C: 2694
Color Temp: 4090 K - Lux: 840 - R: 878 G: 1001 B: 621 C: 2691
Color Temp: 4089 K - Lux: 841 - R: 878 G: 1002 B: 621 C: 2694
Color Temp: 4092 K - Lux: 844 - R: 880 G: 1005 B: 623 C: 2701
```

Figure 9: Fully Lit Room Results

Test Case #2: Fully Dark Room

```
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
Color Temp: 16168 K - Lux: 0 - R: 0 G: 1 B: 1 C: 2
```

Figure 10: Fully Dark Room

Test Case #3: Halfway Lit Room

```
Color Temp: 4028 K - Lux: 411 - R: 439 G: 492 B: 304 C: 1328
Color Temp: 4031 K - Lux: 412 - R: 440 G: 493 B: 305 C: 1330
Color Temp: 4031 K - Lux: 412 - R: 440 G: 493 B: 305 C: 1331
Color Temp: 4030 K - Lux: 413 - R: 440 G: 494 B: 305 C: 1332
Color Temp: 4035 K - Lux: 412 - R: 441 G: 494 B: 306 C: 1334
Color Temp: 4035 K - Lux: 412 - R: 441 G: 494 B: 306 C: 1335
Color Temp: 4035 K - Lux: 412 - R: 441 G: 494 B: 306 C: 1335
Color Temp: 4035 K - Lux: 412 - R: 441 G: 494 B: 306 C: 1335
Color Temp: 4035 K - Lux: 412 - R: 441 G: 494 B: 306 C: 1335
Color Temp: 4037 K - Lux: 414 - R: 442 G: 496 B: 307 C: 1338
```

Figure 11: Halfway Lit Room

Test Case #1: Fully Lit Room with Direct Flashlight

```
Color Temp: 4972 K - Lux: 5327 - R: 4920 G: 6456 B: 4461 C: 17178
Color Temp: 4985 K - Lux: 8414 - R: 8114 G: 10376 B: 7280 C: 27798
Color Temp: 4920 K - Lux: 10292 - R: 10126 G: 12738 B: 8915 C: 34030
Color Temp: 4837 K - Lux: 11757 - R: 11905 G: 14638 B: 10222 C: 39250
Color Temp: 4819 K - Lux: 11786 - R: 12094 G: 14736 B: 10310 C: 39652
Color Temp: 4814 K - Lux: 11120 - R: 11482 G: 13934 B: 9762 C: 37558
Color Temp: 4832 K - Lux: 10801 - R: 11025 G: 13487 B: 9436 C: 36256
Color Temp: 4864 K - Lux: 10660 - R: 10648 G: 13223 B: 9227 C: 35343
Color Temp: 4893 K - Lux: 10639 - R: 10511 G: 13165 B: 9191 C: 35106
Color Temp: 4919 K - Lux: 10743 - R: 10515 G: 13267 B: 9268 C: 35339
Color Temp: 4910 K - Lux: 10977 - R: 10768 G: 13561 B: 9469 C: 36129
Color Temp: 4942 K - Lux: 10297 - R: 9939 G: 12667 B: 8838 C: 33691
```

Figure 12: Fully Lit Room with Direct Flashlight

App Testing with Sensor

Below are screen shots from testing at Avila Beach, CA. The data shown is from a clear day at around noon. We took several readings at various levels of ocean depth and height above the surface. The differences, though slight, in the HydroColor App verses our iOcean app are for two main reasons. The first of which is the fact that HydroColor asks you to point the camera at a very specific angle and cardinal direction, and the fact that is based on a grey scale as a base measurement. Due to the fact that it uses the CCD array of the iPhone camera it needs to have a base to judge off of. Our design however, uses LED's and simply use the voltage created as a proportion of the amount of that color light hitting it, no need for a grey scale base or specific directionality. Secondly is the fact that there is no glass covering over our sensing mechanism. Since we are using the sensor directly, not a camera with a lens we obtain slightly more pure readings as the lens of the iPhone camera dims specifically the blue value slightly. You will see the difference in blue between our readings. All this being said however, the comparison was used due to the accepted accuracy of the HydroColor app. It has been available on the iOS App Store for some time, has several positive reviews and when reading the thesis of the creator of the app we determined it to be valuable to compare our data to his. Not that the RGB values on the HydroColor app are called reflectance, the values we obtain are as well, we kept ours in raw number form as we did not plot our results graphically.

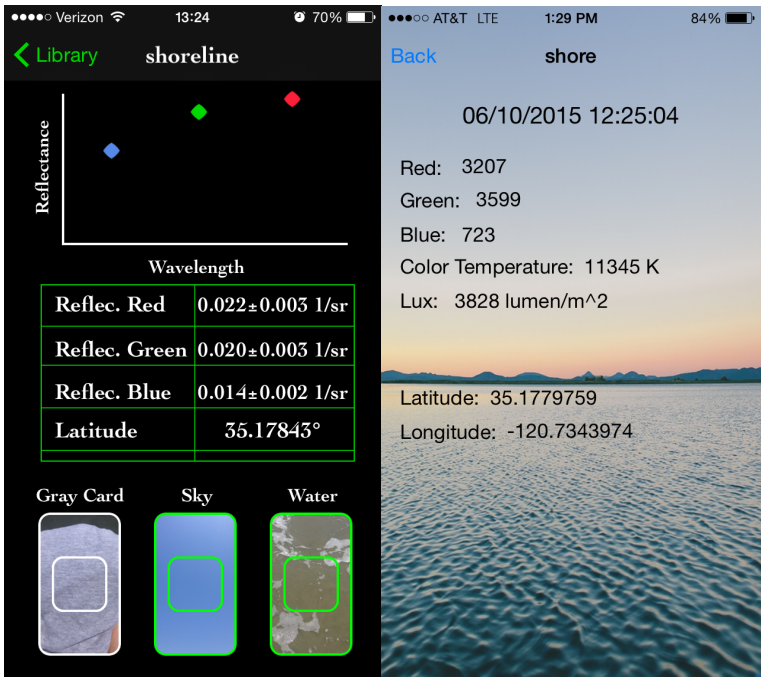

Figure 13: Comparison of Shoreline

As is shown in Figure 13 on both apps when you are just a few feet into the water the blue value is much lower than the green and red values. This was a hard data point to take as there was constantly wave crashing making the water full of foam and whitewash but we were able to capture a good set of comparison data between the apps.

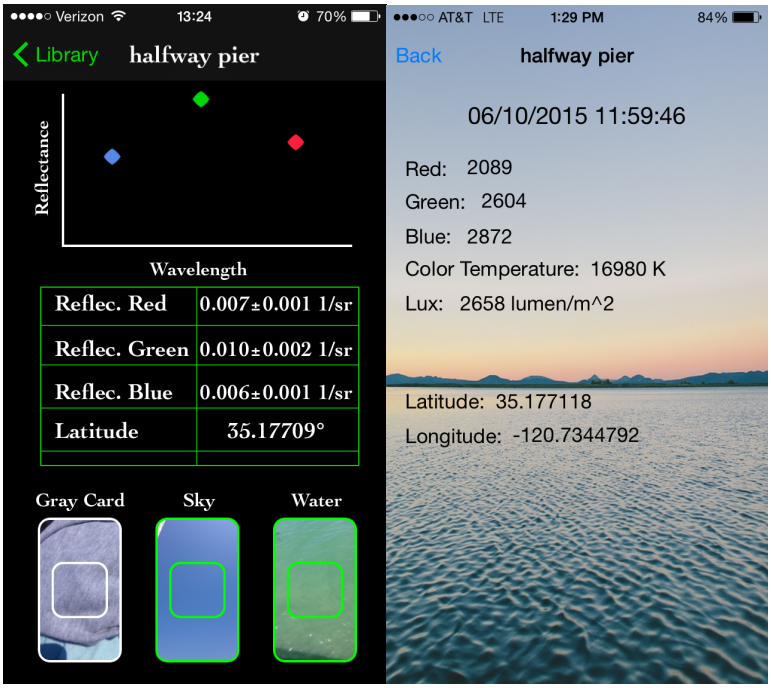

Figure 14: Halfway down the Pier

The values in Figure 14 were skewed mostly due to the difference in direction when collecting data. The HydroColor app uses a compass to make sure you are point the correct direction for its calculations, which led to a green patch of water than where we measured. Although we tried to be as close as possible to the same area, this is the slight difference.

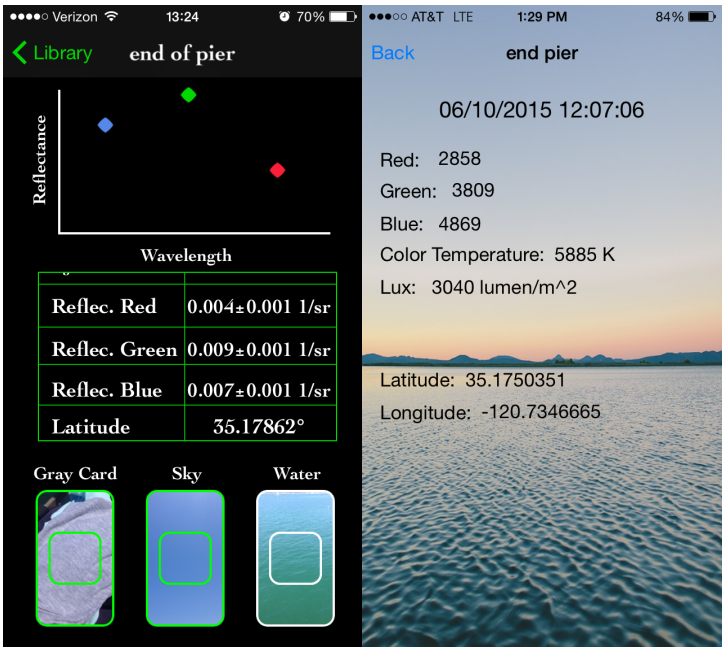

Figure 15: End of the Pier

Again in Figure 15 we see the blue and green values are not the same but we do see that they are much higher in value proportionally to red.

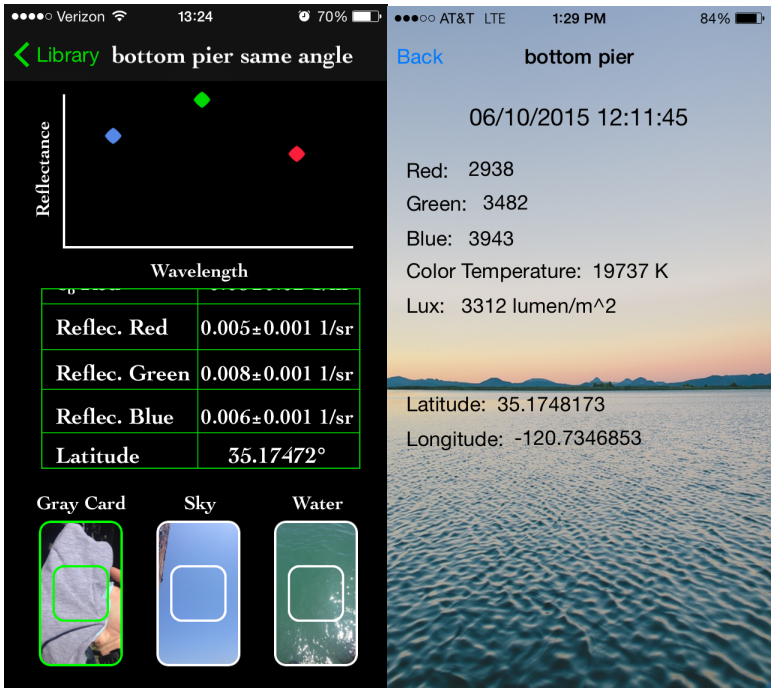

Figure 16: Bottom of the pier

Figure 16 shows that in both images, the red value is the lowest. Even though the blue and green values differ, they are both much higher than the red value.

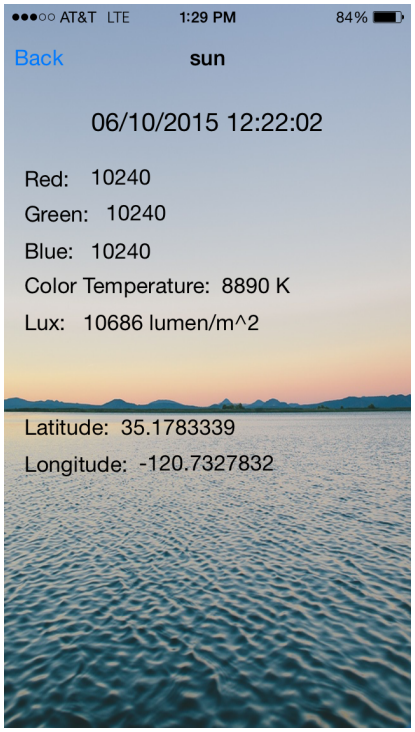

Figure 17: Direct Sunlight

Figure 17 shows the max values for the RGB and lux values, and the warmest temperature in that particular weather.

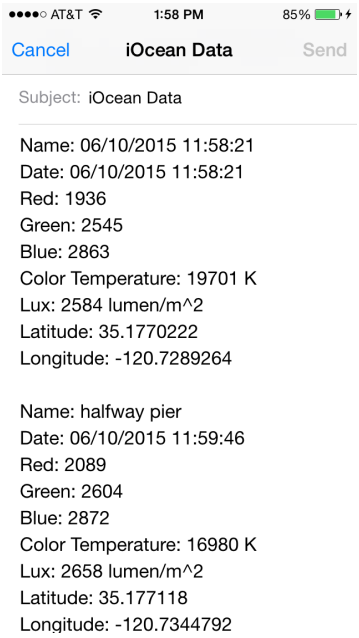

Figure 18: Share Data

Clicking the "Share" button will allow the user to export all of the stored data on phone through email. As seen in Figure 18, each value is properly labeled with the value name and/or units.

Test Results and Conclusions

The sensor proved that it was consistent on it's own scale to varying light. The values that were obtained, though proportionally correct could have been calibrated more to match with the HydroColor app. Most importantly the device is handheld and very portable. The Bluetooth connection to the app and the ability to collect, store and send data were all very successful. The future development of this device and the arena of color sensing will lead to a more accurate color space and hopefully a more precise conversion to what data is take to what we perceive as color.

Overall the project is a successful first step in the development of a handheld ocean color sensor that utilizes a smart-phone to save, display, and share the data collected. Though the data is not perfect it was accurate on a raw scale and proportional to itself which

shows the sensor worked correctly and took the readings it was supposed to take. It is not a perfect sensor but it is a very good prototype.

Future Work

There are a number of ways we feel the project could be improved upon. For example, we only used off-the-shelves parts when implementing our design. If we had more time, we could design parts more centric to our design, which would make the hardware more accurate and tuned to specifically ocean sensing, not just color in general. Also, if we used higher resolution diodes, then the raw data would be more precise and accurate. A more robust microprocessor would also be able to take and manipulate data much faster.

References

Board of Directors. "IEEE IEEE Code of Ethics." *IEEE.org*. IEEE, 25 Nov. 2013. Web. 23 Oct. 2014. <http://www.ieee.org/about/corporate/governance/p7-8.html>.

Brown, Owen. "Arduino Tutorial: Integrating Bluetooth LE and IOS with Swift - Ray Wenderlich." Ray Wenderlich. N.p., 15 Nov. 2014. Web. 26 Apr. 2015.

Crowd Sourced. "Adafruit/Adafruit_TCS34725." GitHub. Adafruit, 20 Dec. 2014. Web. 26 Apr. 2015.

Deschamps, Pierre-Yves. "POLDER and Ocean Color." *POLDER and Ocean Color*. CNRS, Laboratoire De Physique Et Chimie Marine, n.d. Web. 25 Oct. 2014. <http://www.ioccg.org/reports/polder/polder.html>

Deschamps, Pierre-Yves, Bertrand Fougnie, Robert Frouin, Pierre Lecompte, and Christian Verwaerde. "SIMBAD: A Field Radiometer for Ocean-color Validation." *SIMBAD: A Field Radiometer for Satellite Ocean-color Validation* (n.d.): n. pag. *Final Frontier*. University of California San Diego, June 2011. Web. 20 Oct. 2014. <http://finalfrontier.ucsd.edu/Public/SIMBADA_Project/>.

Evaluation of Satellite Ocean Color Data Using SIMBADA Radiometers (n.d.): n. pag. *International Ocean Color Coordinating Group*. IOCCG, 11 June 2104. Web. 25 Oct. 2014. <http://www.ioccg.org/groups/OCR-VC/OCR-VC_SIMBADA_sm.pdf>.

Howard, Brian C. "Ocean Health Gets "D" Grade in New Global Index." National Geographic. National Geographic Society, 02 Oct. 2014. Web. 19 May 2015.

James L. Mueller, Giulietta S. Fargion And Charles R. Mcclain, Editors, and J. L. Mueller, R.w. Austin, A. Morel, G.s. Fargion, And C.r. Mcclain, Authors. *NASA/TM-*

2003-21621/Rev-Vol I (n.d.): n. pag. *Ocean Color WEB*. NASA, 05 Sept. 2014. Web. 25 Oct. 2014. <http://oceancolor.gsfc.nasa.gov/>.

James L. Mueller, Giulietta S. Fargion And Charles R. Mcclain, Editors, and J. L. Mueller, R.w. Austin, A. Morel, G.s. Fargion, And C.r. Mcclain, Authors. *NASA/TM-2003-21621/Rev-Vol II* (n.d.): n. pag. *Ocean Color WEB*. NASA, 05 Sept. 2014. Web. 25 Oct. 2014. <http://oceancolor.gsfc.nasa.gov/>.

Laboratoire D'Optique Atmosphérique. "Result Filters." *National Center for Biotechnology Information*. U.S. National Library of Medicine, 10 July 2004. Web. 25 Oct. 2014. <http://www.ncbi.nlm.nih.gov/pubmed/15285097?report=abstract>.

Mobley, Curtis. "Ocean Optics Web Book." Apparent Optical Properties •. Ocean Optics, 3 Feb. 2010. Web. 26 Apr. 2015.

San Diego State University. "Ocean Color Viewed from Space." Ocean Color. San Diego State University, 27 July 1995. Web. 19 May 2015.

Appendix

Shown below are our calculations for data analyzing and our source code.

RGB to Wavelength Calculations

The snippet of code below showed the conversion from RGB to HSV color space and from the Hue we determined a dominant wavelength, however this was inconclusive as there are always infinite wavelengths in each color. We determined to just use lux and RGB instead. However it was a start to a good idea so we decided to save the code.
*Not shown: Sensor collects raw r,g,and b values.

```
cmax = max(max(r, g), b);
cmin = min(min(r, g), b);
luminance = (float)cmax;
delta = cmax-cmin;
R = (float)r/10240.0;
G = (float)g/10240.0;
B = (float)b/10240.0;

if (cmax > 0) {
  saturation = ((float) delta) / cmax;
}
else {
  saturation = 0;
}

if (saturation == 0) {
  hue = 0;
}
else {
  if (cmax == r) {
     hue = (g-b)/delta;
  }
  else if (cmax == g) {
     hue = ((b-r)/delta) + 2;
  }
  else {
     hue = ((r-g)/delta) + 4;
  }
}
hue = hue / 6;
if (hue < 0) {
  hue += 1;
}
//Formulas to extract further data from intrinsic measurements
```

```
lumen = lux * 0.09290304;           //Intensity
lambda = 650-((hue*175)/240);   //Wavelength (approximation)
```

Lux Calculations

The following code snippet shows how the illuminace is calculated

```
float illuminance;

//This only uses RGB ... how can we integrate clear or calculate lux
illuminance = (-0.32466F * r) + (1.57837F * g) + (-0.73191F * b);

return (uint16_t)illuminance;
```

Color Temperature to Qualitative Color Conversion

Color Temperature is defined as the temperature of an ideal black-body radiator that radiates light of comparable hue to that of the light source. This is a very qualitative scale however can help to determine the type of light if it is unknown. In our case we could see what the sensor thought it was being hit with. This was helpful in determining good data from bad. If the color temperature was too low we knew to retake the reading. Table 2 gives the temperature values in Kelvin and corresponding real life examples.

1,700 K	Match flame, low pressure sodium lamps (LPS/SOX)
1,850 K	Candle flame, sunset/sunrise
2,700–3,300 K	Incandescent lamps
3,000 K	Soft (or Warm) White compact fluorescent lamps
3,200 K	Studio lamps, photofloods, etc.
3,350 K	Studio "CP" light
4,100–4,150 K	Moonlight[2]
5,000 K	Horizon daylight
5,000 K	Tubular fluorescent lamps or cool white/daylight compact fluorescent lamps (CFL)
5,500–6,000 K	Vertical daylight, electronic flash
6,200 K	Xenon short-arc lamp[3]
6,500 K	Daylight, overcast
6,500–10,500 K	LCD or CRT screen

15,000–27,000 K	Clear blue poleward sky

Table 2: Color Temperature Scale

TCS34725 Color Temperature conversion:

```
  float X, Y, Z;      /* RGB to XYZ correlation      */
  float xc, yc;       /* Chromaticity co-ordinates   */
  float n;            /* McCamy's formula            */
```

float cct;

```
  /* 1. Map RGB values to their XYZ counterparts.    */
  /* Based on 6500K fluorescent, 3000K fluorescent   */
  /* and 60W incandescent values for a wide range.   */
  /* Note: Y = Illuminance or lux                    */
  X = (-0.14282F * r) + (1.54924F * g) + (-0.95641F * b);
  Y = (-0.32466F * r) + (1.57837F * g) + (-0.73191F * b);
  Z = (-0.68202F * r) + (0.77073F * g) + ( 0.56332F * b);

  /* 2. Calculate the chromaticity co-ordinates      */
  xc = (X) / (X + Y + Z);
  yc = (Y) / (X + Y + Z);

  /* 3. Use McCamy's formula to determine the CCT    */
  n = (xc - 0.3320F) / (0.1858F - yc);

  /* Calculate the final CCT */
  cct = (449.0F * powf(n, 3)) + (3525.0F * powf(n, 2)) + (6823.3F
* n) + 5520.33F;

  /* Return the results in degrees Kelvin */
  return (uint16_t)cct;
```

Arduino C Code

The C code below is divided into three sections: initialization, BLE connection, and data collection/transmission loop. As shown in Figure 7 the primary step is to "find" the sensor. Once this has been accomplished the microprocessor will attempt to connect to another bluetooth device. After the connection has been established the microprocessor will wait for a command to be given by the master bluetooth device and will take data once one is received. This loop will continue as long as the user keeps sending directives to the microprocessor. After a certain amount of time the microprocessor will go into sleep mode in order to reduce power consumption.

Microprocessor and Sensor Code:

```
#include <Adafruit_TCS34725.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/*Hardware Connections
   Connect SCL    to analog 5
   Connect SDA    to analog 4
   Connect VDD    to 3.3V DC
   Connect GROUND to common ground */

//Sensor Parameters:

/* Integration time max counts:
   2.4ms - 1 cycle    - Max Count: 1024
   24ms  - 10 cycles  - Max Count: 10240
   50ms  - 20 cycles  - Max Count: 20480
   101ms - 42 cycles  - Max Count: 43008
   154ms - 64 cycles  - Max Count: 65535
   700ms - 256 cycles - Max Count: 65535 */

/* Gain Options:
  GAIN_1x           = 0x00,   <  No gain
  GAIN_4X           = 0x01,   <  2x gain
  GAIN_16X          = 0x02,   <  16x gain
  GAIN_60X          = 0x03,   <  60x gain */

// Initialize with specific integration time and gain values
Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_24MS, TCS34725_GAIN_4X);
```

```
//Set global variables
uint16_t r, g, b, c, colorTemp, lux;
uint16_t cmax;
uint16_t cmin;
uint16_t delta;
int ack;
int lumen;
int lambda;
float luminance;
float hue;
float saturation;
float R;
float G;
float B;
char *Values[5];
char Red[20];
char Green[20];
char Blue[20];
char Lux[20];
char ColorTemp[20];


// Initialize BLE Module
SoftwareSerial BLE_Shield(4,5); //Rx,Tx

void getData(void) {
    //Take and organize measurements
    tcs.getRawData(&r, &g, &b, &c);
    colorTemp = tcs.calculateColorTemperature(r, g, b);
    lux = tcs.calculateLux(r, g, b) * 2;

    // Experimental Conversion of RBGB values to HSV (Hue, Saturation,
Value)
    // convert RGB (in range 0 to 1) to HSL (in range 0 to 1)
    cmax = max(max(r, g), b);
    cmin = min(min(r, g), b);
    luminance = (float)cmax;
    delta = cmax-cmin;
    R = (float)r/10240.0;
    G = (float)g/10240.0;
    B = (float)b/10240.0;

    if (cmax > 0) {
      saturation = ((float) delta) / cmax;
    }
    else {
      saturation = 0;
    }

    if (saturation == 0) {
      hue = 0;
    }
    else {
      if (cmax == r) {
```

```
            hue = (g-b)/delta;
        }
        else if (cmax == g) {
            hue = ((b-r)/delta) + 2;
        }
        else {
            hue = ((r-g)/delta) + 4;
        }
      }
      hue = hue / 6;
      if (hue < 0) {
        hue += 1;
      }
      //Formulas to extract further data from intrinsic measurements
      lumen = lux * 0.09290304;            //Intensity
      lambda = 650-((hue*175)/240);   //Wavelength (approximation)
}

void array(void) {
  int count = 0;

  itoa(r,Red,10);
  Values[count++] = Red;

  itoa(g,Green,10);
  Values[count++] = Green;

  itoa(b,Blue,10);
  Values[count++] = Blue;

  itoa(colorTemp,ColorTemp,10);
  Values[count++] = ColorTemp;

  itoa(lux,Lux,10);
  Values[count++] = Lux;
}

void setup(void) {
  Serial.begin(9600);         //Set up the serial port at 9600 bps.
  BLE_Shield.begin(9600);     // Set up BLE shield at 9600 bps. This is
the default
                              // BLE baud rate

  if (tcs.begin()) {           // Check to ensure the sensor is
connected
    Serial.println("Found sensor");
  } else {
    Serial.println("No sensor found ... check your connections");
    while (1);
  }
}

void loop(void) {
  if(BLE_Shield.available() && BLE_Shield.read() == 1) {
```

```
    //Set ackknowledge bit to 1
    ack = 1;
    Serial.println(ack, DEC);
    //Give the iPhone the buffer bit
    BLE_Shield.write(" !");
    delay(200);
    //call functions to gather all data
    getData();
    array();
    //Begin sending data to iPhone
    BLE_Shield.write(" ");
    BLE_Shield.write(Red);
    delay(200);
    BLE_Shield.write(" !");
    delay(200);
    while(ack<6) {
      if(BLE_Shield.available()){
        ack = BLE_Shield.read();
        BLE_Shield.write(" ");
        BLE_Shield.write(Values[ack-1]);
        delay(200);
        BLE_Shield.write(" !");
        delay(200);
      }
    }
  }
}
```

iOS Swift Code

The iOS application is developed both programmatically and visually. Swift allows apps to be created through both hard coding and graphical user interface. Figure 8 shows that the app, based on the user's proper input, will connect to the Arduino through bluetooth, save new captures on the mobile device, and display the results. Figure 19 displays all of the different view screens.

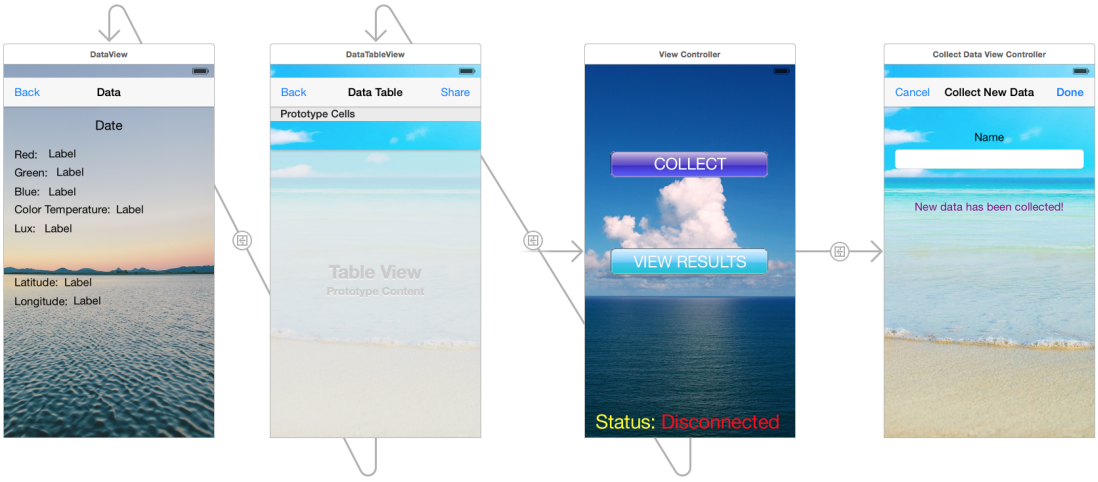

Figure 19. Swift GUI

Code:

```
//
//  BTDiscovery.swift
//  Arduino_Servo
//
//  Created by Owen L Brown on 9/24/14.
//  Copyright (c) 2014 Razeware LLC. All rights reserved.
//

import Foundation
import CoreBluetooth

let btDiscoverySharedInstance = BTDiscovery();

class BTDiscovery: NSObject, CBCentralManagerDelegate {

  private var centralManager: CBCentralManager?
  private var peripheralBLE: CBPeripheral?

  override init() {
    super.init()

    let centralQueue = dispatch_queue_create("com.raywenderlich", DISPATCH_QUEUE_SERIAL)
```

```
    centralManager = CBCentralManager(delegate: self, queue: centralQueue)
  }

  func startScanning() {
    if let central = centralManager {
      central.scanForPeripheralsWithServices([BLEServiceUUID], options: nil)
    }
  }

  var bleService: BTService? {
    didSet {
      if let service = self.bleService {
        service.startDiscoveringServices()
      }
    }
  }

  // MARK: - CBCentralManagerDelegate

  func centralManager(central: CBCentralManager!, didDiscoverPeripheral peripheral:
CBPeripheral!, advertisementData: [NSObject : AnyObject]!, RSSI: NSNumber!) {
    // Be sure to retain the peripheral or it will fail during connection.

    // Validate peripheral information
    if ((peripheral == nil) || (peripheral.name == nil) || (peripheral.name == "")) {
      return
    }

    // If not already connected to a peripheral, then connect to this one
    if ((self.peripheralBLE == nil) || (self.peripheralBLE?.state ==
CBPeripheralState.Disconnected)) {
      // Retain the peripheral before trying to connect
      self.peripheralBLE = peripheral

      // Reset service
      self.bleService = nil

      // Connect to peripheral
      central.connectPeripheral(peripheral, options: nil)
    }
  }

  func centralManager(central: CBCentralManager!, didConnectPeripheral peripheral:
CBPeripheral!) {

    if (peripheral == nil) {
      return;
    }

    // Create new service class
    if (peripheral == self.peripheralBLE) {
      self.bleService = BTService(initWithPeripheral: peripheral)
    }

    // Stop scanning for new devices
    central.stopScan()
  }

  func centralManager(central: CBCentralManager!, didDisconnectPeripheral peripheral:
CBPeripheral!, error: NSError!) {

    if (peripheral == nil) {
```

```
      return;
    }

    // See if it was our peripheral that disconnected
    if (peripheral == self.peripheralBLE) {
      self.bleService = nil;
      self.peripheralBLE = nil;
    }

    // Start scanning for new devices
    self.startScanning()
  }

  // MARK: - Private

  func clearDevices() {
    self.bleService = nil
    self.peripheralBLE = nil
  }

  func centralManagerDidUpdateState(central: CBCentralManager!) {
    switch (central.state) {
    case CBCentralManagerState.PoweredOff:
      self.clearDevices()

    case CBCentralManagerState.Unauthorized:
      // Indicate to user that the iOS device does not support BLE.
      break

    case CBCentralManagerState.Unknown:
      // Wait for another event
      break

    case CBCentralManagerState.PoweredOn:
      self.startScanning()

    case CBCentralManagerState.Resetting:
      self.clearDevices()

    case CBCentralManagerState.Unsupported:
      break

    default:
      break
    }
  }

}

//
//  BTService.swift
//  Arduino_Servo
//
//  Created by Owen L Brown on 10/11/14.
//  Copyright (c) 2014 Razeware LLC. All rights reserved.
//

import Foundation
import CoreBluetooth

/* Services & Characteristics UUIDs */
let BLEServiceUUID = CBUUID(string: "025A7775-49AA-42BD-BBDB-E2AE77782966")
```

```
let PositionCharUUID = CBUUID(string: "A9CD2F86-8661-4EB1-B132-367A3434BC90") //RX
let WriteCharUUID = CBUUID(string: "F38A2C23-BC54-40FC-BED0-60EDDA139F47") //TX
let BLEServiceChangedStatusNotification = "kBLEServiceChangedStatusNotification"


class BTService: NSObject, CBPeripheralDelegate {
    var peripheral: CBPeripheral?
    var positionCharacteristic: CBCharacteristic?
    var writeCharacteristic: CBCharacteristic?

    init(initWithPeripheral peripheral: CBPeripheral) {
        super.init()

        self.peripheral = peripheral
        self.peripheral?.delegate = self
    }

    deinit {
        self.reset()
    }

    func startDiscoveringServices() {
        self.peripheral?.discoverServices([BLEServiceUUID])
    }

    func reset() {
        if peripheral != nil {
            peripheral = nil
        }

        // Deallocating therefore send notification
        self.sendBTServiceNotificationWithIsBluetoothConnected(false)
    }

    // Mark: - CBPeripheralDelegate

    func peripheral(peripheral: CBPeripheral!, didDiscoverServices error: NSError!) {
        let uuidsForBTService: [CBUUID] = [PositionCharUUID]
        let uuidsForBTService1: [CBUUID] = [WriteCharUUID]

        if (peripheral != self.peripheral) {
            // Wrong Peripheral
            return
        }

        if (error != nil) {
            return
        }

        if ((peripheral.services == nil) || (peripheral.services.count == 0)) {
            // No Services
            return
        }

        for service in peripheral.services {
            if service.UUID == BLEServiceUUID {
                peripheral.discoverCharacteristics(uuidsForBTService, forService: service
as! CBService)
                peripheral.discoverCharacteristics(uuidsForBTService1, forService:
service as! CBService)
            }
        }
    }
```

```
    func peripheral(peripheral: CBPeripheral!, didDiscoverCharacteristicsForService
service: CBService!, error: NSError!) {
        if (peripheral != self.peripheral) {
            // Wrong Peripheral
            return
        }

        if (error != nil) {
            return
        }

        for characteristic in service.characteristics {
            if characteristic.UUID == PositionCharUUID {
                self.positionCharacteristic = (characteristic as! CBCharacteristic)
                peripheral.setNotifyValue(true, forCharacteristic: characteristic as!
CBCharacteristic)

                // Send notification that Bluetooth is connected and all required
characteristics are discovered
                self.sendBTServiceNotificationWithIsBluetoothConnected(true)
            }
            else if characteristic.UUID == WriteCharUUID {
                self.writeCharacteristic = (characteristic as! CBCharacteristic)
                peripheral.setNotifyValue(true, forCharacteristic: characteristic as!
CBCharacteristic)

                // Send notification that Bluetooth is connected and all required
characteristics are discovered
                self.sendBTServiceNotificationWithIsBluetoothConnected(true)
            }
        }
    }

    // Mark: - Private

    func writePosition(position: UInt8) {
        // See if characteristic has been discovered before writing to it
        if self.writeCharacteristic == nil {
            return
        }

        // Need a mutable var to pass to writeValue function
        var positionValue = position
        let data = NSData(bytes: &positionValue, length: sizeof(UInt8))
        self.peripheral?.writeValue(data, forCharacteristic: self.writeCharacteristic,
type: CBCharacteristicWriteType.WithResponse)
    }

    func readPosition() -> NSString? {
        // See if characteristic has been discovered before writing to it
        if self.positionCharacteristic == nil {
            return nil
        }

        // readValue function
        self.peripheral?.readValueForCharacteristic(self.positionCharacteristic)

        if ((self.positionCharacteristic?.value) != nil) {
            return NSString(data: self.positionCharacteristic!.value, encoding:
NSUTF8StringEncoding)
        }
```

```
        return nil
    }

    func sendBTServiceNotificationWithIsBluetoothConnected(isBluetoothConnected: Bool) {
        let connectionDetails = ["isConnected": isBluetoothConnected]

NSNotificationCenter.defaultCenter().postNotificationName(BLEServiceChangedStatusNotifica
tion, object: self, userInfo: connectionDetails)
    }

}

//
//  AppDelegate.swift
//  OceanProject
//
//  Created by Archit Mendiratta on 4/16/15.
//  Copyright (c) 2015 Archit Mendiratta. All rights reserved.
//

import UIKit
import CoreData

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?



    func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This
can occur for certain types of temporary interruptions (such as an incoming phone call or
SMS message) or when the user quits the application and it begins the transition to the
background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down
OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate
timers, and store enough application state information to restore your application to its
current state in case it is terminated later.
        // If your application supports background execution, this method is called
instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive state;
here you can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application
was inactive. If the application was previously in the background, optionally refresh the
user interface.
    }
```

```
    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate.
See also applicationDidEnterBackground:.
        self.saveContext()
    }

    // MARK: - Core Data stack

    lazy var applicationDocumentsDirectory: NSURL = {
        // The directory the application uses to store the Core Data store file. This
code uses a directory named "com.xxxx.ProjectName" in the application's documents
Application Support directory.
        let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
inDomains: .UserDomainMask)
        return urls[urls.count-1] as! NSURL
        }()

    lazy var managedObjectModel: NSManagedObjectModel = {
        // The managed object model for the application. This property is not optional.
It is a fatal error for the application not to be able to find and load its model.
        let modelURL = NSBundle.mainBundle().URLForResource("OceanModel", withExtension:
"momd")!
        return NSManagedObjectModel(contentsOfURL: modelURL)!
        }()

    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator? = {
        // The persistent store coordinator for the application. This implementation
creates and return a coordinator, having added the store for the application to it. This
property is optional since there are legitimate error conditions that could cause the
creation of the store to fail.
        // Create the coordinator and store
        var coordinator: NSPersistentStoreCoordinator? =
NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
        let url =
self.applicationDocumentsDirectory.URLByAppendingPathComponent("OceanProject.sqlite")
        var error: NSError? = nil
        var failureReason = "There was an error creating or loading the application's
saved data."
        if coordinator!.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil,
URL: url, options: nil, error: &error) == nil {
            coordinator = nil
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's
saved data"
            dict[NSLocalizedFailureReasonErrorKey] = failureReason
            dict[NSUnderlyingErrorKey] = error
            error = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
            // Replace this with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You
should not use this function in a shipping application, although it may be useful during
development.
            NSLog("Unresolved error \(error), \(error!.userInfo)")
            abort()
        }

        return coordinator
        }()

    lazy var managedObjectContext: NSManagedObjectContext? = {
        // Returns the managed object context for the application (which is already bound
to the persistent store coordinator for the application.) This property is optional since
there are legitimate error conditions that could cause the creation of the context to
fail.
        let coordinator = self.persistentStoreCoordinator
        if coordinator == nil {
```

```
            return nil
        }
        var managedObjectContext = NSManagedObjectContext()
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
        }()

    // MARK: - Core Data Saving support

    func saveContext () {
        if let moc = self.managedObjectContext {
            var error: NSError? = nil
            if moc.hasChanges && !moc.save(&error) {
                // Replace this implementation with code to handle the error
appropriately.
                // abort() causes the application to generate a crash log and terminate.
You should not use this function in a shipping application, although it may be useful
during development.
                NSLog("Unresolved error \(error), \(error!.userInfo)")
                abort()
            }
        }
    }
}


//
//  DataViewController.swift
//  OceanProject
//
//  Created by Archit Mendiratta on 4/30/15.
//  Copyright (c) 2015 Archit Mendiratta. All rights reserved.
//

import UIKit

class DataViewController: UIViewController {
    @IBOutlet weak var navBar: UINavigationBar!


    @IBOutlet weak var dateLabel: UILabel!
    @IBOutlet weak var rLabel: UILabel!
    @IBOutlet weak var gLabel: UILabel!
    @IBOutlet weak var bLabel: UILabel!
    @IBOutlet weak var waveLenLabel: UILabel!
    @IBOutlet weak var luxLabel: UILabel!
    @IBOutlet weak var latLabel: UILabel!
    @IBOutlet weak var longLabel: UILabel!


    @IBAction func backButton(sender: AnyObject) {
        dismissViewControllerAnimated(true, completion:nil)
    }


    var index: Int = 0


    override func viewDidLoad() {
        super.viewDidLoad()
        self.navBar.setBackgroundImage(UIImage(), forBarMetrics: UIBarMetrics.Default)
        self.navBar.shadowImage = UIImage()
        self.navBar.translucent = true
        self.navBar.topItem?.title = snapshots[index].valueForKey("name") as? String


        dateLabel.text = snapshots[index].valueForKey("date") as? String
        rLabel.text = snapshots[index].valueForKey("r") as? String
        gLabel.text = snapshots[index].valueForKey("g") as? String
        bLabel.text = snapshots[index].valueForKey("b") as? String
```

```
        waveLenLabel.text = snapshots[index].valueForKey("waveLen") as? String
        luxLabel.text = snapshots[index].valueForKey("lux") as? String
        latLabel.text = snapshots[index].valueForKey("latitude") as? String
        longLabel.text = snapshots[index].valueForKey("longitude") as? String


        // Do any additional setup after loading the view.
    }


    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}


//
//  ViewController.swift
//  OceanProject
//
//  Created by Archit Mendiratta on 4/16/15.
//  Copyright (c) 2015 Archit Mendiratta. All rights reserved.
//


import UIKit
import CoreData
import CoreLocation


var connected = false


var timerTXDelay: NSTimer?
var allowTX = true
var rxCount: UInt8 = 1
var rxMax: UInt8 = 5
var currData: NSString? = nil
var isNextDataGood = false
var r: NSString! = "0"
var g: NSString! = "0"
var b: NSString! = "0"
var waveLen: NSString! = "0"
var lux: NSString! = "0"
var latitude : NSString! = "N/A"
var longitude : NSString! = "N/A"


var snapshots = [NSManagedObject]()


class ViewController: UIViewController, CLLocationManagerDelegate {
    @IBOutlet var connectLabel: UILabel!
    let locationManager = CLLocationManager()


    let loader = UIAlertController(title: "Please Wait", message: "Collecting data...",
preferredStyle: UIAlertControllerStyle.Alert)


    @IBAction func collectButton(sender: AnyObject) {
        if connected {
            self.locationManager.startUpdatingLocation()
            self.fetchData()
        }
        else {
            var alert = UIAlertController(title: "Unable to collect data", message:
"Power the hardware", preferredStyle: UIAlertControllerStyle.Alert)
            alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.Default,
handler: nil))
            self.presentViewController(alert, animated: true, completion: nil)
            println("Disconnected")
```

```
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

            // Watch Bluetooth connection
            NSNotificationCenter.defaultCenter().addObserver(self, selector:
Selector("connectionChanged:"), name: BLEServiceChangedStatusNotification, object: nil)

            // Start the Bluetooth discovery process
            btDiscoverySharedInstance


        if connected {
            connectLabel.text = "Connected"
            connectLabel.textColor = UIColor.greenColor()
        }
        else {
            connectLabel.text = "Disconnected"
            connectLabel.textColor = UIColor.redColor()
        }

        self.locationManager.delegate = self
        self.locationManager.desiredAccuracy = kCLLocationAccuracyBest
        self.locationManager.requestWhenInUseAuthorization()

        // Do any additional setup after loading the view, typically from a nib.
    }

    func locationManager(manager: CLLocationManager!, didUpdateLocations locations:
[AnyObject]!) {
        CLGeocoder().reverseGeocodeLocation(manager.location, completionHandler: {
(placemarks, error) -> Void in

            if error != nil {
                println("Error: " + error.localizedDescription)
            }

            if placemarks.count > 0 {
                let pm = placemarks[0] as! CLPlacemark
                self.displayLocationInfo(pm)
            }
        })
    }

    func displayLocationInfo(placemark: CLPlacemark) {
        self.locationManager.stopUpdatingLocation()
        latitude = "\(placemark.location.coordinate.latitude)"
        longitude = "\(placemark.location.coordinate.longitude)"
    }

    func locationManager(manager: CLLocationManager!, didFailWithError error: NSError!) {
        println("Error: " + error.localizedDescription)
    }

    func connectionChanged(notification: NSNotification) {
        // Connection status changed. Indicate on GUI.
        let userInfo = notification.userInfo as! [String: Bool]

        dispatch_async(dispatch_get_main_queue(), {
            // Set image based on connection status
            if let isConnected: Bool = userInfo["isConnected"] {
                if isConnected {
```

```
                connected = true
                self.connectLabel.text = "Connected"
                self.connectLabel.textColor = UIColor.greenColor()
                println("Connected to device");
            } else {
                connected = false
                self.connectLabel.text = "Disconnected"
                self.connectLabel.textColor = UIColor.redColor()
                println("Disconnected from device")
            }
        }
    });
}

func fetchData() {
    // 1
    if !allowTX {
        return
    }

    if let bleService = btDiscoverySharedInstance.bleService {
        self.presentViewController(loader, animated: true, completion: nil)
        bleService.writePosition(1)

        // 5
        // Start delay timer
        allowTX = false
        rxCount = 1;
        if timerTXDelay == nil {
            timerTXDelay = NSTimer.scheduledTimerWithTimeInterval(0.1,
                target: self,
                selector: Selector("timerTXDelayElapsed"),
                userInfo: nil,
                repeats: false)
        }
    }
}

func sendPosition(position: UInt8) {
    // 4
    // Send position to BLE Shield (if service exists and is connected)
    if let bleService = btDiscoverySharedInstance.bleService {
        bleService.writePosition(position)

        // 5
        // Start delay timer
        if timerTXDelay == nil {
            timerTXDelay = NSTimer.scheduledTimerWithTimeInterval(0.1,
                target: self,
                selector: Selector("timerTXDelayElapsed"),
                userInfo: nil,
                repeats: false)
        }
    }
}

func assignData(index: UInt8, data: NSString) {
    if (index == 1) {
        r = data
    }
    else if (index == 2) {
        g = data
    }
    else if (index == 3) {
        b = data
    }
```

```
        else if (index == 4) {
            waveLen = data
            waveLen = "\(waveLen) K"
        }
        else if (index == 5) {
            lux = data
            lux = "\(lux) lumen/m^2"
        }
    }

    func readPosition() {

        // 4
        // Send position to BLE Shield (if service exists and is connected)
        if let bleService = btDiscoverySharedInstance.bleService {
            currData = bleService.readPosition()
            //println(currData)

            if currData == " " || currData == nil {
                //ignore
                //println("Got the blank for \(rxCount)")
            }
            else if currData == "!" || currData == " !" {
                isNextDataGood = true;
                //println("Got the ! for \(rxCount)")
            }
            else if isNextDataGood {
                if let data = currData {
                    println(data)
                    assignData(rxCount, data: data)
                }
                isNextDataGood = false

                rxCount++
                sendPosition(rxCount)
                if (rxCount > rxMax) {
                    rxCount = 1
                    allowTX = true
                    loader.dismissViewControllerAnimated(true, completion: nil)
                    let controller =
storyboard?.instantiateViewControllerWithIdentifier("CollectDataID") as!
CollectDataViewController
                    presentViewController(controller, animated: true, completion: nil)
                }

            }

            // 5
            // Start delay timer
            if timerTXDelay == nil {
                timerTXDelay = NSTimer.scheduledTimerWithTimeInterval(0.1,
                    target: self,
                    selector: Selector("timerTXDelayElapsed"),
                    userInfo: nil,
                    repeats: false)
            }
        }
    }

    func timerTXDelayElapsed() {
        self.stopTimerTXDelay()
        self.readPosition()
    }

    func stopTimerTXDelay() {
```

```
        if timerTXDelay == nil {
            return
        }

        timerTXDelay?.invalidate()
        timerTXDelay = nil
    }

    /*override func viewDidAppear(animated: Bool) {

    }*/

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}

//
//  DataTableViewController.swift
//  OceanProject
//
//  Created by Archit Mendiratta on 4/26/15.
//  Copyright (c) 2015 Archit Mendiratta. All rights reserved.
//

import UIKit
import CoreData
import MessageUI

class DataTableViewController: UIViewController, UITableViewDelegate,
MFMailComposeViewControllerDelegate {

    var subject: String = "iOcean Data"
    var body: String = ""
    var next: Int = 0

    @IBOutlet weak var navBar: UINavigationBar!

    @IBOutlet weak var dataTable: UITableView!

    @IBAction func backButton(sender: AnyObject) {
        dismissViewControllerAnimated(true, completion:nil)
    }

    @IBAction func shareButton(sender: AnyObject) {
        let mailComposeViewController = configuredMailComposeViewController()

        if MFMailComposeViewController.canSendMail() {
            self.presentViewController(mailComposeViewController, animated: true,
completion: nil)
        }
        else {
            self.showSendMailErrorAlert()
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.navBar.setBackgroundImage(UIImage(), forBarMetrics: UIBarMetrics.Default)
        self.navBar.shadowImage = UIImage()
        self.navBar.translucent = true
```

```
        // Do any additional setup after loading the view.
    }

    override func viewDidAppear(animated: Bool) {
        dataTable.reloadData()
    }

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)

        //1
        let appDelegate =
        UIApplication.sharedApplication().delegate as! AppDelegate

        let managedContext = appDelegate.managedObjectContext!

        //2
        let fetchRequest = NSFetchRequest(entityName:"Snapshot")

        //3
        var error: NSError?

        let fetchedResults =
        managedContext.executeFetchRequest(fetchRequest,
            error: &error) as? [NSManagedObject]

        if let results = fetchedResults {
            snapshots = results
        } else {
            println("Could not fetch \(error), \(error!.userInfo)")
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return snapshots.count
    }

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath)
-> UITableViewCell {

        let cell = UITableViewCell(style: UITableViewCellStyle.Default, reuseIdentifier:
"Cell")

        cell.textLabel?.text = snapshots[indexPath.row].valueForKey("name") as? String
        cell.backgroundColor = UIColor.clearColor()

        var myCustomSelectionColorView = UIView()
        myCustomSelectionColorView.backgroundColor = UIColor.greenColor()
        cell.selectedBackgroundView = myCustomSelectionColorView

        return cell
    }

    func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {
```

```
        if editingStyle == UITableViewCellEditingStyle.Delete {
            let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
            let managedContext = appDelegate.managedObjectContext!
            managedContext.deleteObject(snapshots.removeAtIndex(indexPath.row) as
NSManagedObject)
            managedContext.save(nil)
            dataTable.reloadData()
        }
    }

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath:
NSIndexPath) {
        var cell:UITableViewCell = tableView.cellForRowAtIndexPath(indexPath)!
        next = indexPath.row
        self.performSegueWithIdentifier("jumpToData", sender: self)
    }

    // MARK: - Navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if segue.identifier == "jumpToData" {
            if let destination = segue.destinationViewController as? DataViewController {
                destination.index = next
            }
        }
    }

    func getSnapshots() -> String {
        var index = 0;
        var rtn: String = ""
        var temp: String

        while (index < snapshots.count) {
            temp = (snapshots[index].valueForKey("name") as? String)!
            rtn += "Name: \(temp)\n"
            temp = snapshots[index].valueForKey("date") as! String
            rtn += "Date: \(temp)\n"
            temp = snapshots[index].valueForKey("r") as! String
            rtn += "Red: \(temp)\n"
            temp = snapshots[index].valueForKey("g") as! String
            rtn += "Green: \(temp)\n"
            temp = snapshots[index].valueForKey("b") as! String
            rtn += "Blue: \(temp)\n"
            temp = snapshots[index].valueForKey("waveLen") as! String
            rtn += "Color Temperature: \(temp)\n"
            temp = snapshots[index].valueForKey("lux") as! String
            rtn += "Lux: \(temp)\n"
            temp = snapshots[index].valueForKey("latitude") as! String
            rtn += "Latitude: \(temp)\n"
            temp = snapshots[index].valueForKey("longitude") as! String
            rtn += "Longitude: \(temp)\n\n"
            index++
        }

        return rtn
    }

    func configuredMailComposeViewController() -> MFMailComposeViewController {
        let mailComposerVC = MFMailComposeViewController()
        mailComposerVC.mailComposeDelegate = self
        mailComposerVC.setSubject(subject)
        body = getSnapshots();
        mailComposerVC.setMessageBody(body, isHTML: false)

        return mailComposerVC
```

```
    }

    func showSendMailErrorAlert() {
        let sendMailErrorAlert = UIAlertView(title: "Could not send the email", message:
"Your device could not send the email. Please check email configuration and try again.",
delegate: self, cancelButtonTitle: "OK")
        sendMailErrorAlert.show()
    }

    func mailComposeController(controller: MFMailComposeViewController!,
didFinishWithResult result: MFMailComposeResult, error: NSError!) {

        switch result.value {

        case MFMailComposeResultCancelled.value:
            println("Cancelled mail")
        case MFMailComposeResultSent.value:
            println("Mail sent")
        default:
            break
        }

        self.dismissViewControllerAnimated(true, completion: nil)
    }

    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a little preparation
before navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        // Get the new view controller using segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */

}

//
//  CollectDataViewController.swift
//  OceanProject
//
//  Created by Archit Mendiratta on 4/26/15.
//  Copyright (c) 2015 Archit Mendiratta. All rights reserved.
//

import UIKit
import CoreData

class CollectDataViewController: UIViewController {



    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var navBar: UINavigationBar!

    @IBAction func cancelButton(sender: AnyObject) {
        dismissViewControllerAnimated(true, completion:nil)
    }

    @IBAction func doneButton(sender: AnyObject) {
        var todaysDate:NSDate = NSDate()
        var dateFormatter:NSDateFormatter = NSDateFormatter()
```

```
        dateFormatter.dateFormat = "MM/dd/yyyy HH:mm:ss"
        var dateInFormat:String = dateFormatter.stringFromDate(todaysDate)

        let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
        let managedContext = appDelegate.managedObjectContext!
        let entity =  NSEntityDescription.entityForName("Snapshot",
inManagedObjectContext: managedContext)
        let snapshot = NSManagedObject(entity: entity!, insertIntoManagedObjectContext:
managedContext)

        if textField.text != "" {
            snapshot.setValue(textField.text, forKey: "name")
        }
        else {
            snapshot.setValue(dateInFormat, forKey: "name")
        }
        snapshot.setValue(dateInFormat, forKey: "date")
        snapshot.setValue(r, forKey: "r")
        snapshot.setValue(g, forKey: "g")
        snapshot.setValue(b, forKey: "b")
        snapshot.setValue(waveLen, forKey: "waveLen")
        snapshot.setValue(lux, forKey: "lux")
        snapshot.setValue(latitude, forKey: "latitude")
        snapshot.setValue(longitude, forKey: "longitude")

        var error: NSError?
        if !managedContext.save(&error) {
            println("Could not save \(error), \(error?.userInfo)")
        }
        snapshots.append(snapshot)
        dismissViewControllerAnimated(true, completion:nil)
    }

    /*@IBAction func add(sender: AnyObject) {
        if textField.text != "" {
            data.append("\(textField.text) r: \(r) g: \(g) b: \(b) wavelength: \(waveLen)
lux: \(lux)")
            NSUserDefaults.standardUserDefaults().setObject(data, forKey: "data")
            self.performSegueWithIdentifier("toDataTable", sender: self)
        }
    }*/

    override func viewDidLoad() {
        super.viewDidLoad()

        self.navBar.setBackgroundImage(UIImage(), forBarMetrics: UIBarMetrics.Default)
        self.navBar.shadowImage = UIImage()
        self.navBar.translucent = true

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a little preparation
before navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
```

```
        // Get the new view controller using segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */

}
```