# Sensing and Visualizing the Deep Blue Ocean Project

Pedro João Loureno Ribeiro
pedro.j.ribeiro@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

**Abstract**

The widespread availability of computing technology is creating opportunities for interventions to promote environmental sustainability and ecological consciousness on the part of existing or potential technology users and new ones. These are domains where the political and cultural contexts have wider implications, which are just starting to be explored by the scientific community. This project aims at applying low-cost sensing technologies to the oceanic Twilight Zone using fishermen and their apparatus as vehicles for deploying new acoustic and image sensors for exploring the deep Ocean. The project will involve adapting existing platforms (such as soundtrap.io and mataki.org) for effective research deployment in the Atlantic Ocean. The project involves deploying and testing the sensor in real-world conditions in collaboration with the Natural History Museum of Funchal in Madeira Islands. The project will use low-cost hardware development kits (https://store.particle.io/) for prototyping.
**Keywords:** Biologging, Electronic Tags, IoT, Mobile Phone Application, Server

## 1. Introduction

In an age marked by the conquest of the space we still know so little about our oceans. They are the lifeblood of planet Earth and humankind. They hold 97% of the planets water and they occupy almost three-quarters of the planet. They support the greatest biodiversity of the planet and they are also one of the largest carbon reservoirs holding 54 times more carbon than the atmosphere. Exploring the ocean is a really big challenge because the technologies that currently exists are really expensive, which limits the amount of work that can be done. On the other hand, with the emergence of the climate changes, it has become increasingly necessary to explore and understand the impact that pollution will have on the lifestyle of the marine animals and the ocean itself.

As we know we are moving towards an ubiquitous world where the technology will blend in with the environment and become almost invisible. In recent years, we have seen an exponential increase in the number of small computers, that made possible new discoveries in a variety of areas due to their relative low cost, small size and adaptability to almost every circumstance. As Mark Weiser, principal scientist and manager of the Computer Science laboratory at Xerox Parc, wrote in a magazine.[27]

> *The goal is to achieve the most effective kind of technology that which is essentially invisible to the user. To bring computers to this point while retaining their power will require radically new kinds of computers of all sizes and shapes to be available to each person. I call this future world Ubiquitous Computing.*

Nowadays we are living in a trend where "the growing availability of computing technologies such as mobile GPS enabled devices, capable of image capturing and processing, ecology and environmental science are now capitalizing on the talents and geographical spread of non-specialists citizens, with spare time, curiosity and a smart phone". [23, 14] This is allowing members of the general public, typically non scientists, to generate a new type of data collection and analysis that can be called Citizen Science. The amount and diversity of current citizen science projects is astonishing. We have volunteers monitoring the night sky for light pollution, different type of birds, free divers capturing the extent of oceanic pollution etc. With this new citizen science projects we can expect an exponential growth on the data available which in the long term can help us to better understand what surround us.

In this paper we describe a system that is capable of monitoring the oceans with the use of cheap microcontrollers and at the same time provide almost to real time data to to both biologists and users. In chapter 2 we introduce some concepts like biologging that are a base of this system. We also talk about some microcontrollers that are currently available in the market. In chapter 3 we talk about how we implemented the project and why we took

some decisions. In chapter 4 we demonstrate the tests that we conducted and their respective results, we conclude on the chapter 5 and also present some future work ideas that can be done to improve the system.

## 2. Related Work

In this Section we analyze some existing relevant systems that made us take some decisions while building our solution. We also talk about some topics that are needed to understand our project.

### 2.1. Citizen Science and Biologging

In introduction we talked a little about a new trend called citizen science. Citizen science is a movement that begun recently with the increasing availability of cheap devices, such as microcontrollers, and the fact that the computational power that we carry in our pockets today is absurdly larger than what we had a few years ago. The benefits are assumed to extend beyond the production of large databases. Many people argue that participants will increase their understanding about the process of science through the engagement in authentic science.[23, 25]

There are currently many research programs which monitor and identify bats, bees, birds, flies and slugs for example, but citizen scientists don't need to work outdoors. Non-specialists have been trained to scan satellite imagery for wildebeests in the Serengeti, penguins in the Antarctic, and African migrant groups affected by environmental catastrophes who require emergency aid.[14]

As we know studding wild animals can present major challenges, so in order overcome them biologging technology started to appear. Biologging can be described as the use of miniaturized animal-attached tags for logging and/or relaying of data about an animal's movements, behavior, physiology and/or environment. Biologging technology substantially extends our abilities to observe, and take measurements from, free-ranging, undisturbed subjects, providing much scope for advancing both basic and applied biological research.[24]

In the last 20 years we have seen a rapid development in technology and miniaturization which made possible that biologging could be applied to almost every animal even the ones that live in harsh environments for humans.[16] This has led to particularly rapid advancements in the marine realm where direct observations are almost impossible.

Electronic tagging of marine animals is increasingly being used by scientist to track their movements. These tags allow us to see where and how marine animals travel, and then we can correlate with the ocean environment and understand what drives their movements.[9, 17]

Currently there are a few types of electronic tags that try to meet the needs that different types of observations need, but they are far from being perfect. If we look at Archival tags they are small data loggers that can record swimming depth, internal and external temperatures and ambient light levels. They can record data for up to 10 years depending on the battery and the tag's sampling frequency. However in order to download the data they need to be retrieved, so they can only be used in species that have a high likelihood of being recaptured. There are other types of tags that communicate by GPS with the Argos satellite network.[11] Usually the tags that use this system, pop-up satellite archival tags and satellite positioning tags, are extremely expensive and so, they can only be used by groups that have enough funding and in species that spend enough time on the surface so that the antennas can be outside the water. GSM tags are another type of tags that uses GSM connection to send the data, but because of this they can only be used in coastal waters where there is signal. On the other hand they aren't as expensive as the ones that communicate with the Argos satellite network.

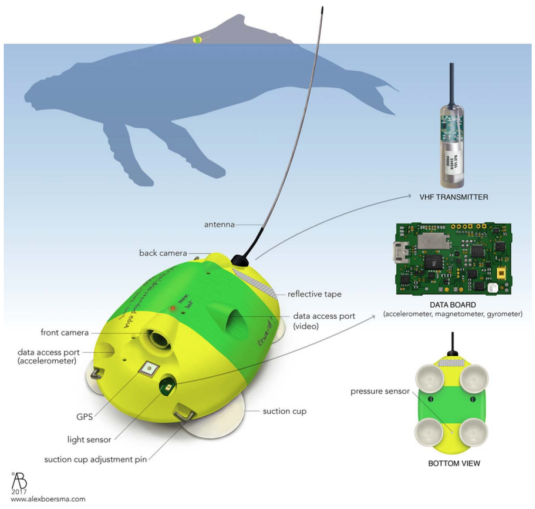


Figure 1: Scheme of a Digital tag.[21]

Digital acoustic recording tags Figure: 1 are the ones being used to monitor marine mammals since they are attached to them by suction cups, and they detach themselves from the animal when their life time is coming to an end. This kind of tags are the closest ones to our idea since we want to utilize and study the same kind of animals.

Recently a couple of projects using affordable microprocessors started to appear. Raspiwhale [18] is a project similar to the one we are going to develop that utilizes a low cost microprocessor equipped to dive to great depths in the sea aboard a whale.

### 2.2. Microcontrollers

A microcontroller is a small computer on a single integrated circuit. It has a processor, memory for storage, RAM to run the programs and for data input and output interfaces. The main difference between a microcontroller and a microprocessor is that the first one has all the structure and components necessary for computing in a single chip, while the other uses external memory for program and data storage.

Microcontrollers are designed to perform specific tasks. For example, keyboards, mouse, washing machine, pen drive, etc. Since the applications are very specific, they need small resources like RAM, ROM, I/O ports etc and then they can be embedded on a single chip, which leads to reduced sizes and costs. With the development of this kind of chips nowadays we can find a lot of this microcontrollers with everything that we need like Wi-Fi, GPS, GSM, etc.[22]

Table 1: - Comparison between the principals features of the three boards considered in this project.

| | Particle Photon | ADAFRUIT HUZZAH | ARDUINO UNO |
|---|---|---|---|
| CLOCK SPEED | 120MHz | 80MHz | 16MHz |
| FLASH MEM | 1MB | 4MB | 32KB |
| DIGITAL PINS | 18 | 9 | 14 |
| ANALOG PINS | 6 | 1 | 8 |
| ANTENNA | PCB and uFl | PCB | No |
| BATTERY | No | Yes | No |
| ONLINE SERV. | Yes | Yes | No |
| Cost(USD) | $19 | $16 | $25 |

In the Table 1 is possible to observe the comparison between the three proposed microcontrollers. Since we are aiming at a modular device with the cheapest price possible all this microcontrollers fulfill this requirement. The Adafruit Huzzah is the one that has the most flash memory and built-in battery, yet is the one with the smallest amount of digital and analog pins. Comparing now the Particle Photon with the Arduino Uno, we can see that both of them have an equivalent number of pins, however the Photon has much more memory, a greater clock speed and an antenna. The only disadvantage is that it does not come with the ability to connect the battery directly to the microcontroller, requiring a regulator.

## 3. Implementation

In this chapter we will describe the implementation decisions made to go from the architecture design and planning to the final version of our proposal.

### 3.1. Requirements

The widespread availability of computing technology is creating opportunities for interventions to promote environmental sustainability and ecological consciousness. Taking into account the main goal of our project we define the following set of requirements that our solution must fill:

- **Affordable** We want this solution to be available to the biggest number of people possible, so that it can be used in citizen science projects. For that to append we need it to be as cheap as possible.
- **Close to Real Time Data** The biggest difference from our solution to the previous ones, is that we want to provide data more often, as close to real time as possible.
- **Modular System** To be used in citizen science projects we want to allow everyone to choose what kind of information they want to retrieve from the devices, allowing them to easily add and remove their own sensors.
- **Public Access API** Citizen science projects usually allow everyone to see the data collected. To continue with this approach we defined an API that allows everyone to retrieve the data from almost any type of device.
- **Battery Life** Last but not least, we want to maximize the duration of the time that the devices are recording data. Since this device mainly usage is going to be ocean monitoring, we wont have energy available, so it is crucial to maximize the battery duration.

### 3.2. Architecture

The Architecture of this project is the combination of three different components. The device, the server and the mobile application.

For the device we opt out for the Particle Photon, since we based some of the sound recording code from Vasconcelos thesis.[26] Another reason that made us choose this option was the fact that everyone in our department was using the same device, so it also made sense for us to use it.

One of the objectives of this work was to build a system that was as modular as possible. In order to

simulate a real world system this project includes some environmental sensors Table: 2.

Table 2: - List of sensors included in the device.

| Sensors | Output | Ports |
|---|---|---|
| GPS | GPS coordinates | Rx-Tx |
| Bar100 | Temp., Pressure, Depth, Alt. | I2C |
| Turbidity Sensor | Turbidity | D6 |
| pH Sensor | pH | A2 |
| Dissolved Oxygen | Oxygen | A5 |
| Microphone | Audio Recording | A1 |

The device only talks with the user application where it sends the information that has stored in the SD card, when requested. In order to facilitate the communication an API was created. This API needs to take into account some limitations that exists in the Particle Photon. In order to create a Wi-Fi network we need to use the device in listening mode by using the library SoftAP HTTP from particle [7]. This library creates a temporary access point and an HTTP server on port 80. The main objective for this library is to allow the users to connect to the device, and give them the ability to configure the Wi-Fi access points that they want the device to attempt to connect. This library also allows the system to provide HTTP URL so that applications can add their own pages to the SoftAP HTTP server. Nevertheless this mode creates some limitations in terms of the size that the requests can have. After some tests conducted the sweet spot where we don't lose any information when answering a request is around 5KB. This raises a problem, because if the information in the file to send is larger than these 5KB, we need more than one request to obtain the information. At first we wanted to send everything in the JSON format to create the same API that we have on the server, but due to this limitation the alternative created was to save the data in CSV format, which is a way to condensate the information in comparison with the JSON format. Now, since we need a way to tell the client when all the information was received and what was the latest information that they received, we need to include two additional fields. the "dataPart" field, that tells the user what was the latest byte of information that they received from the file saved in the SD card, and the "isFinalData" field, that as the name suggests, tells the user if there is more information to send. In order to avoid a repetition in the information sent to the user we also added a field called "deviceId" that identifies from which device the user is getting the data.

Due to the limitation in the size of the responses we decided that the audio recordings can't be send to clients because it will take to much time to send them all, and so we decided to just record and save them in the SD card. Later this information can be processed offline for any purpose.

The mobile application is implemented in iOS since is the platform that i'm most used to develop in. This application is the link between the device and the server since we don't have internet connection at the sea to allow direct communication between them. Since nowadays most of the communications between applications and servers are using HTTP requests, it made sense for us to also use them. Another advantage of this choice is the fact that they allow for the creation of a public API that makes the development of client and server independent. The mobile application also stores the data locally so that it can work offline and send it to the server. In order to achieve this we implemented a database using Realm [8]. Realm is the fastest and the most efficient data base currently in iOS. It's also a multiple platform solution that is very simple to implement.

Lastly, for the server we choose to use Node.js [19] with a REST API for two main reasons. The first one is the fact that our requests are JSON objects, which in reality are javascript objects, the language that Node.js uses, so choosing node for the server makes the server really simple. The last reason was the fact that at the proposal of this thesis professor Nuno Nunes already told that we were going to use Node.js for the server. Also by using a REST API we are separating the client from the server, which makes it possible to develop the various areas of the project independently. It's also possible to develop multiple types of clients that use the same API to retrieve the information.

The server API has two types of requests. A GET request to enable the retreat of information and a PUT request so that it is possible to put information to be kept persistently.

In order to save the data in a persistent way, we choose a NoSQL Database mongoDB [5]. Since in this project the data is actually recordings of sensors at a given time there is no need for a relational database. By choosing a NoSQL database and in particularly mongoDB [5] we gain numerous advantages such as:

1. Non-relational and schema-less data model
2. Low latency and high performance
3. Highly scalable

4. Object-oriented programming that is easy to use and flexible

### 3.3. Implementation

In order to make it easier to understand the implementation description, we will talk about the components in separate like in the architecture section.

Starting by the device, while turned on it can be at three different modes. The first one and the most basic is the idle mode. Due to the fact that the device needs at the start an internet connection to set the current time on the RTC, this mode is the initial one. To set up the time we use the SparkTime library[15]. This library creates a simple NTP client that allows the RTC to synchronize it's clock. The timezone that we selected to the current project is the UTC since it's a uniform standard time for legal commercial and social purposes.

The second mode is the recording mode. In this mode the device records audio and the values from the sensors. The audio recordings are constant, which means that is always recording and saving the files in chunks of 4 seconds per file. The values from the sensors only get recorded if the timer defined by the user is surpassed. For the audio recording we based on the version developed by Dinarte Vasconcelos [26] with a few changes to the code, because we don't send the information to the server and instead we save it in the sdCard.

Since nowadays there is a wide variety of sensors, and we want this device to support all of them, while keeping it simple to add and remove sensors, we decided to enforce that every sensor must implement an interface called *SensorsInterface*, Listing 1. This *SensorsInterface* is an interface with only two methods. A *record()* method that is used to record the value and keep it in a variable at the choice of the user and a *getRecordValue()* method that returns a String with the value that the user wants to save. With this implementation we force the user to create a *class* for every sensor which implies that the code for that sensor will only live inside that class, which makes the application independent from the sensors, and the sensors independent from themselves.

```
class SensorsInterface  {
  public:

// record the value from the sensor
    virtual void record() = 0;

// gets the value from the sensor
    virtual String getRecordValue() = 0;
};
```

Listing 1: SensorsInterface.

Although this interface is very simple, it is actually very powerful if it's used correctly. This interface is the main reason of why it is so easy to add and remove sensors. In our config file there is a parameter called *csvInitialString* where the users should specify the beginning of the CSV file. The users must then respect the order of this string when getting the sensors values.

Last but not least, we have the SoftAP mode. While in this mode the device creates an Wi-Fi hotspot where everyone can connect and get the data using HTTP requests.

```
void myPage(const char* url,
    ResponseCallback* cb, void* cbArg,
    Reader* body, Writer* result, void*
     reserved);
```

Listing 2: softAP handler declaration.

The softAP mode when activated runs on a different thread than the main program execution, that's why we have to take into account the concurrency problems that might occur. Another big problem is that always enabling the Wi-Fi also increases the battery consumption, which is one thing that we want to avoid. The reason why we cannot be in the record and softAP mode at the same time, is because in recording mode we are constantly accessing the SD Card to save the audio recordings, so to avoid conflicts while accessing it, the two modes can't work together. The Listing 2 shows us a declaration of an handler *myPage*. This handler will then be called every time a client makes an HTTP request on the URL *http://192.168.0.1/*. The *url* argument in the function is the path of the file requested by the client, and doesn't include the server name or port. So in our application the typical *url* will be */data?0*. The *cb* argument is a response callback, and it's used by the application to indicate the type of HTTP response, 200(OK) or 404(not found). The *cbArg* is data that should be passed as the first parameter to the callback function. The body is an object that the page handler uses to retrieve the HTTP request body. The result is a writer object that the handler uses to write the HTTP response body. The last argument *reserved* is there for future expansion of the softAP library and for now always equals to *nullptr* [7].

Our server uses the Express framework [1] in order to facilitate the creation of the REST API. In the Listing 3 we can see the beginning of our server application where the constant *app* is the express object that is defined in the file *./app.js*. Using the HTTP built in Node.js we start the server on the port 8080, or in the port given by the cloud provider.

```
const http = require('http');
const app = require('./app');

const port = process.env.PORT || 8080;

const server = http.createServer(app);
```

```
server.listen(port);
```

Listing 3: HTTP Server.

We defined a *mongoose* schema that can be seen in Listing 4 with all the fields for the sensors information and their respective types.. Mongoose is a mongoDB object modeling for Node.js and allows us to save our data to the database and retrieve it. The *id* field is a concatenation of the deviceId and the date that allow us to create a unique identifier for every device recording across the server, mobile application and device. The date is saved in ISO8601 [13] format as a string. We use this format to standardize so that every application can convert to and from it, and once more allowing the server to be independent from the data.

```
const mongoose = require("mongoose");

const dataSchema = mongoose.Schema({
  _id: String,
  deviceId: String,
  date: String,

  latitude: Number,
  longitude: Number,

  temperature: Number,
  depth: Number,
  altitude: Number,
  pressure: Number,
  turbidity: Number,
  ph: Number,
  oxygen: Number
});

module.exports = mongoose.model("data",
    dataSchema);
```

Listing 4: Data model schema.

Right now we have a simple API allowing only a *GET* and *POST* request, and in the *GET* request we always send all the available data. If required by the client applications this API can be easily extended, and for example only send information about one specific device given the respective *deviceId*. Even though on this project we didn't implement any security measures, such as integrity of the data so that it couldn't be changed by unknown entities, the server always check if the data received supports the *dataSchema* presented above. If not it's immediately rejected by the server.

For the mobile application development we followed the MVC pattern. This pattern is widely used for developing user interfaces that divides an application into three interconnected parts. In iOS we have three types of objects with different roles: model, view, controller[12]. Each one of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. In figure 2

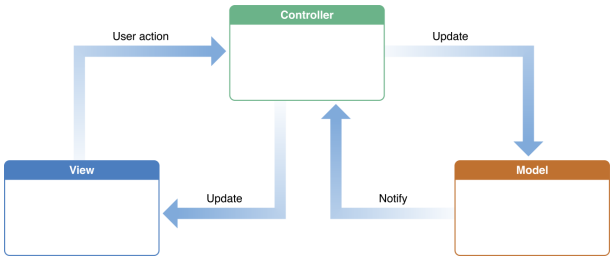


Figure 2: MVC example.

we can see an example of how the communication is handled using this separation of concepts.

Since we are using Realm, and realm forces us to develop a model of the object that we want to save, it made sense for us to use this model across the entire of our application, and not only to save the information. In Listing 5 we can see the optional initialization used when receiving data from both the server and the devices. This initialization makes it really simple to change the data received, for example, if we remove or add one sensor.

```
class DataModel: Object {

    /// init of the object used when
        receiving from server and device
    convenience init(isFromServer: Bool,
        deviceId: String, date: String,
        latitude: Double, longitude: Double
        , temperature: Double, depth:
        Double, altitude: Double, pressure:
         Double, turbidity: Double, ph:
        Double, oxygen: Double) {
        self.init()

        self.id = deviceId + date
        self.isFromServer = isFromServer

        self.date = DateFormatter().
            stringToDate(str: date)!
        self.deviceId = deviceId

        self.latitude = latitude
        self.longitude = longitude

        self.temperature = temperature
        self.depth = abs(depth)
        self.altitude = altitude
        self.pressure = pressure
        self.turbidity = turbidity
        self.ph = ph
        self.oxygen = oxygen

        isNull = false
    }
```

Listing 5: DataModel schema object.

In terms of the mobile application interface we developed four different screens to help us mimic a real world case. In Figure 3 we can see how the users can move inside the application. All the views are inside a *UIPageViewController* that implements the swipe gestures to allow the application to go to

the next page. In order to facilitate the users can also touch one of the buttons in the bottom of the screen to move between pages. The first screen is just a welcome page that allows the user to connect to a nearby device, if the mobile phone is in the device network.


Figure 3: Screens images from left to right: first, second, third and fourth screen

In the second screen we were inspired by the visualization Nightingales Rose [6] and since there was no library available for iOS that could do that, we decided to implement it ourselves. In order to draw everything we used the UIKIT framework [10] from apple. Our design solution starts by dividing a circle between the number of items that we want to place. For example a year will have 12 semi circles, one for each month, so we need to divide the circle in 12 equal parts. The easiest way to achieve this is to divide the degrees of a circle, 360 degrees, by the number of parts we want. This division will give us an angle that we can use to draw. Thankfully the UIKIT framework has *UIBezierPath* class that allows us to draw lines on the screen. This class has a method that creates an arc (Listing: 6) and we then just need to add a line from the arc to the center of the circle and close the path, that it will automatically connect the other side of the arc to the last point of the line, which is the center of the circle. And by creating this simple object we have a semi circle. We then do exactly the same thing to every other semi circle that we want to draw.

```
let path = UIBezierPath()

path.addArc(withCenter: CGPoint(x:
    bounds.midX, y: bounds.midY),
    radius: radius, startAngle:
    startAngle, endAngle: endAngle,
    clockwise: true)
path.addLine(to: CGPoint(x: bounds.midX
    , y: bounds.midY))
path.close()
```

Listing 6: UIBezierPath method addArc.

The only think that varies between the circle is the radius and the start and end angles of the arcs. In order to achieve this we need to scale all the data with the information that we have. The user needs to take car of the labels and we need to have one label for each semi circle to draw. To facilitate and standardize the way to send the data to the visualization we created a structure *CrimeaRoseData*, that can be seen in the Listing 7. This structure makes possible to the visualization to receive any number of different elements at the same time. In our case we will only use two dates, represented by the colors blue and yellow, but the visualization can handle any number.

```
struct CrimeaRoseData {
    var arrayOfData = [Double]()
    var color = UIColor()

    init(arrayOfData: [Double], color:
        UIColor) {
        self.arrayOfData = arrayOfData
        self.color = color
    }
}
```

Listing 7: CrimeaRoseData.

One key aspect of our view is that it is responsive to the user so it can handle a few movements. We added four gesture recognizer. The *UIPinchGestureRecognizer* lets the user zoom in and out in the view allowing for an easy understanding. The *UIPanGestureRecognizer* allows the user to move the image in the screen, which is useful when combined with the pinch to zoom gesture. Then we have the *UIRotationGestureRecognizer* that makes possible the rotation of the view, and last but not least we have the *UITapGestureRecognizer* that recognizes the semi circle that is touched and shows their values. To complement the visualization we added a segmented control to allow the user to choose what type of data they want as well as a date type picker so that the user can select in what temporal order they want to compare the data. In the bottom of the screen we added two buttons that allow the user to select the dates that they want to compare. When they are touched it appears a pop up with a date time picker that allows the users to select the date. There is also two labels that tell the user what date they choose and the correspondent color. Bellow this there are two other labels that show to the user the values of the semi circles when they touch them.

For the third screen we choose to use a library *charts* [20] that allow us to freely customize the chart we want to draw. Every chart drawn with this library has right away animations, dragging, panning and scaling. In order to implement this graphic in a view we instantiated an object of the type *LineChartView* and added it as a subview of our main view. The *LineChartView* as an attribute data of the type *ChartData* that needs to be set in order to display the information. In our case since

we are drawing in a line chart we used the object *LineChartData* which extends the *ChartData*. The user has the possibility to choose if they want to see the data by day, week, month or year by selecting the option in the segmented control at the top of the screen. If the user scroll down they can see numerous switches. Each switcher makes visible in the graphic the data to the correspondent attribute. Because of restrictions with the size of the screen we only feel comfortable in showing at maximum three attributes at the same time, so if the user selects another one it automatically deselects one of which was being shown. To select the date the user has to click on the arrows bellow the segmented control and the view will automatically refresh and display the new data.

In the forth screen we used the apple MapKit framework [4] to visualize the GPS coordinates recorded by the device. For each location recorded we added a *MKAnnotation* which creates a pin in the map. The map automatically aggregates the pins if they are to close to each other, and shows them when the user zooms in. We created a pop up that appears when the user touches a pin, and gives all the information about that recording like the time, location and sensors data. In order to create this pop up we create a *PopupLauncher* object that has a *collectionView* which in turn has any given number of cells. This cells are just a model object that contains a name of an icon and the value to show.

## 4. Results

Our work comprises the development of a system that could sustain the adverse conditions of the oceans, while maintaining the requirements described on the section 3. Due to the difficulty in deploying the device in the ocean we opt to put it on the Tejo river near Santarm. For safety reasons we didn't submerge the case, we left it floating on the river, and because of this we didn't test the mechanism of switching between the recording mode and the SoftAP mode since the water level was always above one meter.

### 4.1. Hardware Prototype

As already said, in order to simulate a real environment we installed some sensors in an particle photon device. The list of the sensors can bee seen in the table 2. We used a peli 1150 case [2] Figure 4 as a box containing all the sensors, power bank and the device. The case by itself is already waterproof but since there are some sensors that need to be outside the case we had to drill some holes. The turbidity sensor needs to be in direct contact with the water and doesn't have a large cable so we did a hole in the bottom of the case just for this


Figure 4: Exterior of the prototype case.

sensor. This sensor can be seen in the fig. 4 as the transparent sensor in the bottom of the case. The Dissolved Oxygen and the pH sensors can be connected with a cable so we made two holes on the top of the case that allow us to connect the cables, and if we are not using them we can simply detach them. The Bar100, the red sensor in Figure 4, has a long enough cable that allows the hole to be on the top of the case while the sensor still reaches the water. In order to power the device we placed a 10000 mhA powerbank inside the case. All the other sensors are inside the box and don't have any contact with the exterior.

### 4.2. Device Requests

In order to test how would the device behave in a real situation, were we can have multiple people getting the same information at the same time, we conducted some tests. We choose the software jmeter [3] from Apache to conduct those tests. Jmeter gives us the ability to do load and performance tests on many different applications. In our case we are interested in the ability to test the requests from the device. In the first test we choose to simulate a single user doing 10 requests to simulate a user getting a large file of data. In the Figure 5 we can see that in average a request takes about 600 milliseconds and for a total of 10 requests it took 6 seconds to get all the answers.

For the next test we tried to simulate two concurrent users each making 10 requests, obtaining a 1 second of time for each request. In spite of the fact that the time for each request doubled, we can see other problems if we look closer to the Figure 6. In order to better simulate a real world application we made some restrictions to the requests. The biggest one is the timeout of 2 seconds for each request. With this restriction we can see that in the column status two of the requests didn't arrive since the status isn't ok. This results look good if

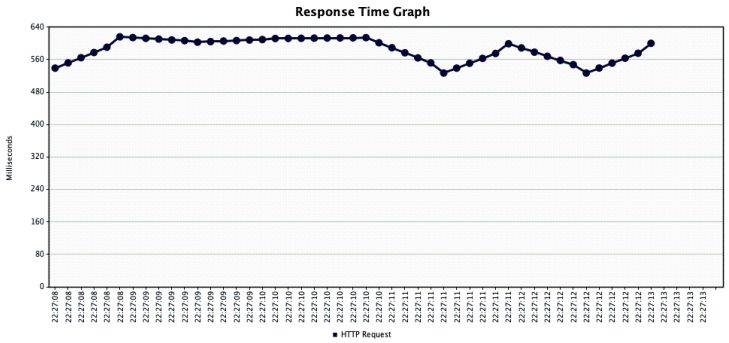


Figure 5: Response time graph for 10 requests made by a single user.

we don't look to the bytes column where it shows us the actual number of bytes that we received. Since we know that the length of the answer is about 4 kB we can assume that a correct answer should have the number of bytes equal to 4133. With this information we can see that half of the requests made aren't correct so we can conclude that the device can't handle two users at the same time.

| Sample # | Start Time | Thread Name | Label | Sample Time(ms) | Status | Bytes | Sent Bytes | Latency | Connect Time(... |
|---|---|---|---|---|---|---|---|---|---|
| 16 | 22:40:27.264 | Group 1 1-1 | HTTP Request | 1748 | | 224 | 119 | 1748 | 1171 |
| 15 | 22:40:26.753 | Group 1 1-2 | HTTP Request | 1685 | | 4133 | 119 | 1681 | 1008 |
| 20 | 22:40:30.020 | Group 1 1-1 | HTTP Request | 505 | | 2172 | 0 | 0 | 135 |
| 1 | 22:40:20.381 | Group 1 1-1 | HTTP Request | 535 | | 4133 | 119 | 533 | 7 |
| 12 | 22:40:25.105 | Group 1 1-1 | HTTP Request | 1049 | | 4133 | 119 | 1046 | 7 |
| 2 | 22:40:20.882 | Group 1 1-2 | HTTP Request | 647 | | 4133 | 119 | 554 | 3 |
| 9 | 22:40:24.056 | Group 1 1-2 | HTTP Request | 537 | | 224 | 119 | 537 | 3 |
| 7 | 22:40:23.540 | Group 1 1-2 | HTTP Request | 515 | | 2172 | 0 | 0 | 2 |
| 8 | 22:40:23.010 | Group 1 1-1 | HTTP Request | 1049 | | 4133 | 119 | 1045 | 2 |
| 19 | 22:40:29.492 | Group 1 1-2 | HTTP Request | 1033 | | 224 | 119 | 1033 | 1 |
| 3 | 22:40:20.917 | Group 1 1-1 | HTTP Request | 1045 | | 4133 | 119 | 1044 | 0 |
| 4 | 22:40:21.530 | Group 1 1-2 | HTTP Request | 955 | | 4133 | 119 | 953 | 0 |
| 5 | 22:40:21.963 | Group 1 1-1 | HTTP Request | 1046 | | 4133 | 119 | 1044 | 0 |
| 6 | 22:40:22.485 | Group 1 1-2 | HTTP Request | 1055 | | 4133 | 119 | 1045 | 0 |
| 10 | 22:40:24.059 | Group 1 1-1 | HTTP Request | 1046 | | 224 | 119 | 1046 | 0 |
| 11 | 22:40:24.593 | Group 1 1-2 | HTTP Request | 1128 | | 224 | 119 | 1128 | 0 |
| 13 | 22:40:25.721 | Group 1 1-2 | HTTP Request | 1031 | | 4133 | 119 | 1024 | 0 |
| 14 | 22:40:26.155 | Group 1 1-1 | HTTP Request | 1108 | | 4133 | 119 | 1102 | 0 |
| 17 | 22:40:28.438 | Group 1 1-2 | HTTP Request | 1053 | | 224 | 119 | 1053 | 0 |
| 18 | 22:40:29.012 | Group 1 1-1 | HTTP Request | 1008 | | 224 | 119 | 1008 | 0 |

Figure 6: Table showing results 10 requests made by two concurrent users.

There are two big reasons for the problems with multiple concurrent users. The first one is the limitations on the photon SoftAP mode that wasn't made for this kind of application. The second one and probably the one that limit's the most is the fact that every request is trying to access the SD card, and every request is on a different thread that we can't control. With this we will have concurrency problems where two users are trying to get information from different parts of the SD card at the same time.

### 4.3. Server Requests

In order to evaluate the server and database performance we conducted a load test for them. The tests were conducted on the same machine of the server, so the address used was *http://localhost:8080/data*. To test we used the jmeter [3] software. We stress test the server with 100 concurrent users each one doing 100 requests. All the requests were answered correctly by the server and the test duration was 1 minute and 14 seconds. To compare we did the same tests but changing the number of concurrent users. The results can be seen in the Table 3

| | 1 User | 50 Users | 100 Users |
|---|---|---|---|
| Samples | 100 | 5000 | 10000 |
| Avg Resp. | 10ms | 380 ms | 730 ms |
| Error % | 0 % | 0 % | 0 % |
| Throughput | 95.3/sec | 127.5/sec | 134.4/sec |
| Received - KB/sec | 1121.8 | 1500.44 | 1581.41 |
| Sent - KB/sec | 11.36 | 15.19 | 16.01 |

Table 3: - Server stress test.

### 4.4. Real World Test Scenario

In order to simulate the real world system we conducted a test where every component of our project works together. The objective of the test was to get all the information into a user that was never close enough to the device to collect the data. In order for this to be possible the information in the device needs to be collected by another user with the use of the mobile application. The information needs then to be sent to the server by this application. To conclude the test the initial user needs to open his mobile application with internet access and get the data from the server. The test succeeds if the information in both users application is the same.

To conduct this test, and since we can't use more than one device to install the application we used a real iPhone 7 as the user that gets the data from the device, and the iPhone X emulator available in macOS as the user that can't get close enough to the device and can only get the information from the server.

In figure 7 we can see the result obtained after performing the simulation test. On the left side we have the iPhone 7, which we used to get the data from the device, while on the right we have the iPhone X which is the one getting the data from the server. We can see that both of them got the same information which leads us to affirm that the test was successful.

## 5. Conclusions

There are no doubts that we are moving into an ubiquitous world. IoT devices are becoming cheaper and consequently they are increasing. They are also becoming more and more powerful which allows us to do things today that were impossible a few years ago. The majority of these IoT devices are made to do simple tasks and communicate through the internet, so they are a really good fit for home automation, collection of meteorological information, etc. With these devices users nowa-

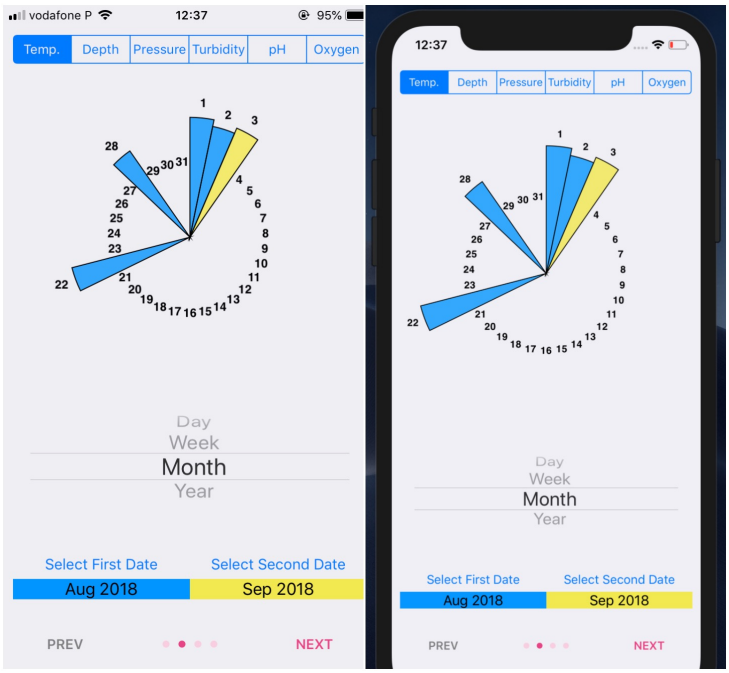


Figure 7: Mobile application after the test. iPhone 7 on the left and iPhone X on the right.

days tend to expect that the information that they are visualizing is in real time, and this aspect makes a lot of difference for the user perspective about the systems.

Nowadays, in ocean exploration, the costs are extremely high which makes this exploration very slow and also only available to groups with lots of resources. On the contrary, in land, we have many people that contribute for example in animal researches, and they do that as a hobby with money from their own pocket. Most of these people that actively contribute to science can't even imagine to do the same in the ocean mostly because of the costs involved, however this whole situation can change with the adaptation of these cheap IoT devices.

In this paper we talk about a new solution to use cheap IoT devices for ocean monitoring that can be easily built, that are adaptable to almost any kind of sensor, and that provides almost to real time data to any user. In order to demonstrate the capabilities of our solution we conducted some performance tests on both the server and the device. We also simulated a real world scenario to prove that our solution can in fact be used in the real world.

### 5.1. Future Work

Since these devices aren't made for this kind of task, we already expected that our solution would be far from perfect. In fact there are a few number of suggestions that may or may not help these kind of systems to become available to almost everyone. Probably the biggest change that can be made is to add LoRaWAN Support, a new technology that supports long range Wi-Fi communication. Since one of the biggest limitations in this project comes from the access point mode on photon, and since particle is currently developing the version 8 of it's software that will allow for tcp connections while on this mode, the restrictions that we are currently facing in the limited size of the responses will stop to exist, which will make possible, for example, to send all the sound recordings to the mobile devices. There is also an huge improvement that can be made on the mobile application interface to make the visualization more appealing to the users. It wasn't a real focus on this project but can be massively explored now that we have a mechanism to get a huge amount of data without expending a lot of money.

## Acknowledgements

First of all, i would like to thank my advisor, professor Nuno Nunes for all the support, availability, and knowledge transmitted.

I also would like to thank to Dinarte Vasconcelos, Miguel Ribeiro and Marko Radeta for the all help and contribution, with both comments and suggestions that allowed me to overcome some obstacles.

A big thank you to my family and in special to my uncle that help me to build the case that was fundamental for the conclusion of the project.

Last but not least i want to thank to my friends for all the support that helped me and motivated me throughout this project.

## References

[1]

[2] 1150 protector case.

[3] Apache jmeter.

[4] Mapkit.

[5] Mongodb for giant ideas.

[6] Nightingale's rose.

[7] Particle.

[8] Realm: Create reactive mobile apps in a fraction of the time.

[9] Scor-tagging.

[10] Uikit.

[11] Worldwide tracking and environmental monitoring by satellite.

[12] Cocoa core competencies, Apr 2018.

[13] Date and time format - iso 8601, Mar 2018.

[14] R. Ashcroft. They walk among us the rise of citizen science, Aug 2016.

[15] Bkobkobko. bkobkobko/sparktime.

[16] S. Bograd, B. Block, D. Costa, and B. Godley. Biologging technologies: new tools for conservation. introduction. *Endangered Species Research*, 10:17, Mar 2010.

[17] S. J. Cooke, J. D. Midwood, J. D. Thiem, P. Klimley, M. C. Lucas, E. B. Thorstad, J. Eiler, C. Holbrook, and B. C. Ebner. Tracking animals in freshwater with electronic tags: past, present and future. *Animal Biotelemetry*, 1(1):119, May 2013.

[18] Daniel. Raspiwhale: First raspberry pi mounted on a whale, Nov 2015.

[19] N. Foundation.

[20] D. Gindi. danielgindi/charts, Sep 2018.

[21] J. Goldbogen, D. Cade, A. Boersma, J. Calambokidis, S. Kahane-Rapport, P. Segre, A. Stimpert, and A. Friedlaender. Using digital tags with integrated video and inertial sensors to study moving morphology and associated function in large aquatic vertebrates. *The Anatomical Record*, 300(11):1935–1941, 2017.

[22] M. Greer. Iot development board comparison, Apr 2016.

[23] M. Radeta, N. J. Nunes, D. Vasconcelos, and V. Nisi. Poseidon - passive-acoustic ocean sensor for entertainment and interactive data-gathering in opportunistic nautical-activities. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, pages 999–1011, New York, NY, USA, 2018. ACM.

[24] C. Rutz and G. C. Hays. New frontiers in biologging science. *Biology Letters*, 5(3):289–292, 2009.

[25] D. J. Trumbull, R. Bonney, D. Bascom, and A. Cabral. Thinking scientifically during participation in a citizen-science project. *Science Education*, 84(2):265–275.

[26] D. G. vasconcelos. *Biodiversity Monitoring Using Smart Acoustic Sensors Application to mosquitos*. PhD thesis, 2017.

[27] M. Weiser. Some computer science issues in ubiquitous computing. *Commun. ACM*, 36(7):75–84, July 1993.