5-2016

Visualizing Aquatic Species Movement with Spatiotemporal Data from Acoustic and Satellite Transmitters

Perabjoth Singh Bajwa
University of New Orleans

VISUALIZING AQUATIC SPECIES MOVEMENT WITH SPATIOTEMPORAL DATA FROM ACOUSTIC AND SATELLITE TRANSMITTERS

An Honors Thesis Submitted to the Faculty of

The Department of Computer Science of

The University of New Orleans

In Partial Fulfillment of the Requirements for the Degree of

Bachelors of Science,

With Honors in Computer Science

Department of Computer Science

The University of New Orleans

By

Perabjoth Singh Bajwa

May 2016

Acknowledgement

I would like to thank Dr. Mahdi Abdelguerfi for providing me with the opportunity to work on this project under his guidance as well as providing me with the proper resources and equipment to enable this project. I would also like to thank Daniel Ward and Nathan Cooper for their help in developing and optimizing the application. In addition, I would like to thank the Louisiana Department of Wildlife and Fisheries, especially Ashley Ferguson, for providing the necessary resources, manpower and feedback to improve the application. Finally, I would like to thank Dr. Shengru Tu for helping in the completion of this thesis.

Contents

Table of Figures:

Abstract

Tracking an individual specimen can be a difficult task especially when one also has to keep track of the environmental factors that affect the tracked specimen's behavior. The task of tracking these animals becomes impossible when they become submerged in water and their number increases to more than just one. The aquatic species that are being tracked by this project in Lake Pontchartrain and the Gulf of Mexico are: tarpon, scalloped hammerhead, whale shark, tiger shark, yellowfin tuna, spotted seatrout, redfish, and bull shark. We are tracking these fish using acoustic and satellite transmitters. The insertion of transmitters in the fish was handled by the Louisiana Department of Wildlife and Fisheries biologists. The acoustic transmitters were implanted on smaller fish that only swam in Lake Pontchartrain. Due to this, receivers were only implanted at locations across the lake on various types of attachments such as buoys, PVC pipes, and pilings. These receivers were positioned at more than ninety locations in order to maximize the acquisition of detections. These species were tracked in Lake Pontchartrain and the Gulf of Mexico. After this preliminary setup, a constant batch of data was generated on a regular basis and this data was process by the application developed in this project. A Ruby on Rails application was then setup in order to store this data and manipulate it to display an animated track. The application utilizes: Ruby, Rails, HTML, CSS, SQL, JavaScript and multiple third part libraries. Many optimizations were performed in order to ensure reliability and performance when loading a high volume of fish or if a high volume of users were to use the application.

Keywords: Fish, Tracking, Acoustic, Telemetry, Satellite, Rails, Ruby, Web Application, SQL, PostgreSQL.

Introduction

Observing species in their natural habitat is an important part of biological research; however, the study of fish is impeded by the logistics of tracking a single specimen amongst a large region inhabited by many other animals. Acoustic and satellite tagging solve these problems – researchers can observe fish movements remotely and retrieve high-resolution tracking information. In addition, multiple fish may be tracked simultaneously, which helps researchers observe their movements in groups. Previous studies made by the Louisiana Department of Wildlife and Fisheries as a cooperative study with the Louisiana State University at Lake Calcasieu and the University of New Orleans collected large quantities of acoustic data on fish at Bayou St. John. However this data was not fully utilized at the time and was exclusively available to researchers involved with the studies (Louisiana Department of Wildlife and Fisheries, Web).

Telemetry, a successor of these studies, using data collected by the University of New Orleans at Lake Pontchartrain and the Gulf of Mexico, hopes to progress these studies by providing a visual component that can aid in the analysis of the collected data. Telemetry is a PostgreSQL-driven Ruby on Rails application that allows biologists to observe the movements of fish via visual renderings of satellite and acoustic data and to discover movement patterns based on environmental dynamics. Data is parsed into the system and analyzed to produce paths for each participating animal. A mapping component yields a graphic visualization of these paths. Combined with environmental sampling including salinity, lunar cycles, and weather patterns, these tools aim to provide biologists a rich toolset to monitor and analyze fish patterns.

Sampled fish in Lake Pontchartrain are tagged with transmitters emitting signals to stationary hydrophones mounted on buoys, pilings and PVC poles in the lake. Data is collected periodically from the receivers by field agents and imported into the Telemetry system. A user can then fetch this data from the front-end by using the appropriate filtering criteria or an animal specific Web-address. This initiates an AJAX request that retrieves data consisting of detection objects detailing the latitudes, longitudes, and timestamps for each fish. Client-side, this data is stored into JavaScript, which is then preprocessed and sorted chronologically. A simulation is then run from the earliest time in the dataset to the latest. Movement of each fish from one location to another in the visualization is performed based on interpolating each adjacent point. A vector is generated for each animal. The fish move along this vector depending on a set simulation speed until they reach the next detection point. These results are rendered onto a Leaflet map, which uses Esri tiled layers as its base. Esri allows the usage of the ArcGIS mapping services in Leaflet (Jgravois, Web). Each fish is represented using a marker with an image of the corresponding species.

Another version of the tracking service has been compiled into a jQuery function that allows insertion into any other HTML element. This feature is currently being demonstrated at *http://fishla.org/fish-tracker/* and is made available to the public. The features uses an external library (*head.js*) in order to dynamically load the scripts required for the fish visualization to render the fish's paths onto a map. After loading all the necessary scripts, it appends the styling to the head so that the visualizations render properly. Afterwards, all the elements are generated and inserted into the HTML element

that was selected. The mechanism by which the motion of the fish works is implemented in the same manner as the main application; however, numbered markers are used here instead of images. This secondary implementation of the fish tracker allows other developers to integrate the fish tracking service into their own website.

The original fish tracking services are available to the public at *https://louisianafisheries.net/telemetry* in its initial release. The Telemetry project not only allows biologists to monitor fish movements, but also gives access to the fishermen for better planning their fishing strategies. With this tool, both biologists and fishermen will have a better understanding of how different factors affect fish movement patterns.

The Problem

Tracking an individual animal and relating different factors to its movement patterns can be quite difficult. This difficulty exists due to the human aspect that might interfere with regular patterns as the presence of the human might startle the animal and lead to changed behavioral patterns. Moreover, there is an increase in danger to the individual monitoring the creature as the observed specimen or other animals in the same habitat might consider the human as a threat, especially in the case of dangerous species. In addition, the difficulty increases when water is introduced to the equation, as is the case with marine animals. Tracking animals can lead to a variety of positive outcomes. Some of which are obvious such as a direct increase in our general knowledge about the behavioral patterns of that animal. Others not so obvious, such as the conservation of these species of animals, which is true in our case. The majority of recreational anglers mostly catch Spotted Seatrout. Around 85% of these Seatrout are from state water along the Gulf of Mexico, where most (50-60% annually) of it takes place in Louisiana (Callihan, Web). This activity plays an important economical role in the state in terms of its monetary ($757 million in 2006) value as well as its ability to create jobs (7800 jobs) (Callihan). So we can see that preserving these species has a positive cultural, economic and social benefit to the state of Louisiana. Certain species of fish are farmed by the LDWF so that when these species start to run low in the lake, they would release them there in order to preserve the ecological system (Ward, Interview). In addition to the initial difficulty of tracking a single specimen, the ante is increased when the sample size is inflated to more than one animal. Doing such a thing individually is virtually impossible especially in the oceanic realm. Hence, the involvement of technology is

required in order to eliminate any possible dangers and increase the effectiveness of the data collection process.

The Solution

Involving automated technology almost always guarantees greater efficiency, and in our case, it helps make what was once impossible a reality. Using a multitude of technologies, we were able to track multiple fish in Lake Pontchartrain as well as the Gulf of Mexico. The technologies involved are Vemco transmitters and receivers, SPOT tags, a PostgreSQL database, and a Ruby on Rails application, which itself employs other web technologies such as HTML, JavaScript, CSS and SQL.

The Physical Aspect

The use of transmitters and receivers is one of the key components that enabled this study. Louisiana Department of Wildlife and Fisheries (LDWF) have planted receivers throughout Lake Pontchartrain. The receivers consist of an outer cylinder, end cap, a metal/PVC internal casing and a battery compartment, as shown in Figure 1:

Figure 1: A receiver opened to show its constituent components: outer cylinder – end cap – metal/PVC internal casing – battery compartment (Vemco, Web).

These receivers were mounted on different kinds of attachments that vary in size and shape. In Lake Pontchartrain, they are attached to buoys, pilings, and PVC poles at more than ninety sites across the lake. Figure 2 depicts the spread of the receivers in the lake

(Ward, Email).

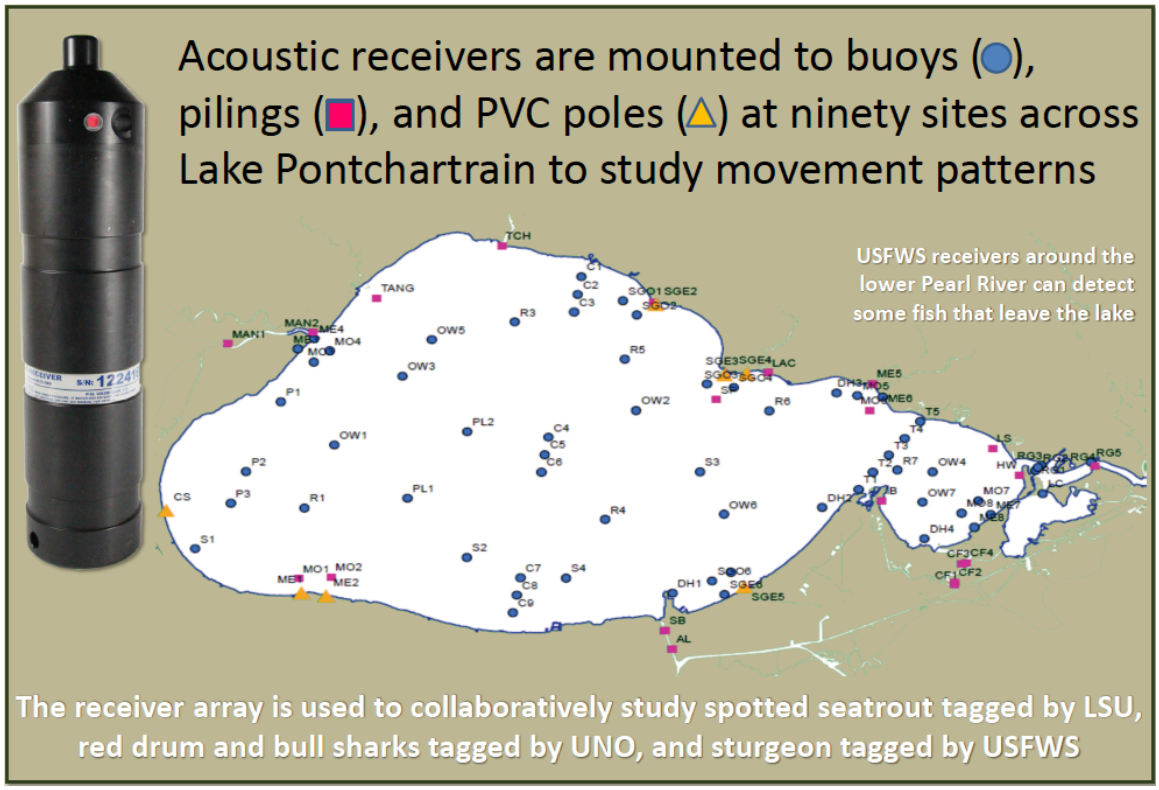

Figure 2: Image of a receiver as well as a map showing the locations of the receivers and the type of attachment they are attached to.

In addition to these receivers, LDWF biologists also implant transmitters in the fish. The attachment of these transmitters is essential to the project because if it is not performed correctly, the fish could lose its transmitter and yield false data about the fish's path. Each fish is captured by a volunteer angler and brought to a surgery vessel, where the biologist places the fish in a holding tank and aerates it with oxygen to relieve the fish from stress. Measurements of the fish are then taken including weight, length, and the dart tag, a blue tag with a phone number and unique ID (Ferguson, E-mail). This tag can be used to track the fish should it ever be captured again. After the measurements are taken, the fish is placed in a surgery cradle (Ferguson). While the fish is on the cradle,

water is passed over its gills and a small incision is made on the ventral side of the fish where an acoustic transmitter is inserted (Ferguson). The size of the transmitter being inserted is selected based on the fish's size so that it doesn't protrude and is easily detectable in the fish's body (Ferguson, Interview). After the insertion is complete, three simple interrupted sutures are made to close the incision (Ferguson, Email). These sutures are loops that are not connected to each other. The fish is placed in a recovery tank where it rests for at least 30 minutes before it is released back in to the lake (Ferguson). Figure 3 shows the process by which the fish receives the acoustic tag:

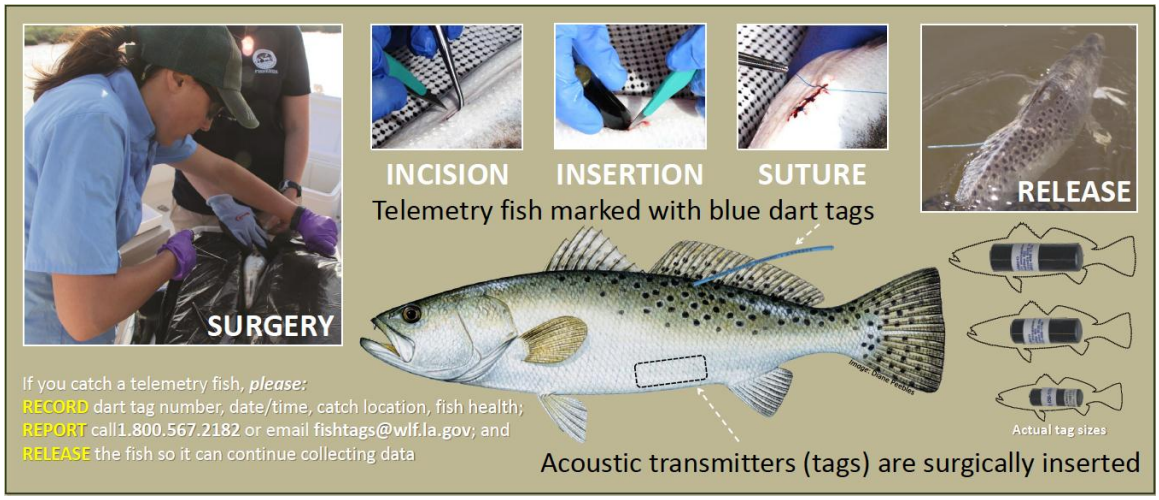

Figure 3: Illustration depicting the surgical process by which a fish receives a transmitter (Ward, Email).

The acoustic transmitters and receivers used on the fish are from Vemco. The frequencies of the acoustic transmitters match those of the receivers so that data from other projects using the same type of technology would not cause any interference. In addition, the receivers are adjusted according to the water depth and kind of attachment. These receivers generate detections that are retrieved wirelessly by the LDWF biologists from the buoys across Lake Pontchartrain using Bluetooth technology (Ferguson, Interview). The satellite detections are retrieved online by authorized users from the

LDWF. All data is then provided to the application in the form of CSV (Comma Separated Value) files with specified headers. These CSV files are then parsed into temporary files that hold only the necessary information for the database. Once all the CSV files are parsed, the data is inserted into the database to the corresponding tables.

The Software

The Model

The database was implemented in PostgreSQL using Rails models and migrations. PostgreSQL was chosen is because it was free, fast and reliable. These three factors insured that the application would operate at no additional monetary, performance or maintenance cost. Moreover, PostgreSQL is capable of handling high volumes of activity and data (The PostgreSQL Global Development Group, Web). This is evident from the three million plus data entries that we are storing and the thousands of users that make use of the service.

Fish and Detections

Our database stores all persistent features the application requires, ranging from buoy locations to fish detections. The way the tables in the database are organized have an impact on how easy it is to retrieve the data. The core table in the database upon which the rest of the data depends on is the fish table. A fish record consists of the following columns: a transmitter id, species id, dart tag, name, weight, total length, time captured, and tag type. The transmitter id or dart tag can be used to uniquely identify the fish. Since

the transmitter communicates with receivers, the next thing we need are the receivers themselves. The receiver table consists of the following columns: a receiver number, and a buoy id. The buoy id in the receiver refers to an attachment known as a "buoy" in this case. The buoy table consists of the following columns: a site id, an attachment, the bottom type and notes. The site id from the buoy is linked to a site whose table consists of the following columns: the site name, longitude and latitude. It can be inferred that from a receiver id we can get a buoy, which will then give us a site that will help us determine the location of the detection. Due to this relationship, the detections table consists of these columns: a receiver id, a transmitter id and the time. However, this relationship relies on the fact that all fish are communicating with acoustic transmitters to hydrophones mounted on attachments. This isn't the case when it comes to larger species that navigate through the Gulf of Mexico. Bigger fish such as Whale Shark, Scalloped Hammerhead and Yellowfin Tuna make use of the SPOT satellite tags, which transmit direct coordinates of these fish along with their timestamps. Hence, another table was created solely for the satellite detections of these fish which consisted of these columns: the dart tag number, latitude, longitude and the time. Although, we can now successfully query for detections that correspond to the fish, we still do not know what kind of fish are being discussed. Hence, we need to link the species id column of the fish to another table. We make a table called species which consists of only one column: the common name. Each species id found in the fish table will now correspond to its correct common name in the species table.

Retrieval of Environmental Data

The basic detections are nice to have in order to display the fish tracks and monitor their movements, but along they give little context to these patterns. For this reason, environmental data such as water salinity, water temperature and tide is desirable. Unfortunately, this data was either not readily available or was not available at a reasonable rate. For example, all three were available from the buoys but only for the time when the biologist would go to collect the fish detection data from the buoys. This means that there would only be a reading of water temperature, tide and salinity every 2-4 weeks, which is not feasible due to its low frequency. For this reason, I tried to find other sources that provide this data at a more reasonable rate. There was no single source that could provide me with all three of these environmental variables. The next best thing would be a source that provided me with at least two of these variables, which was what I found next.

The NOAA's (National Oceanic and Atmospheric Administration) website exposes an API which allows users to request data given a set of parameters (NOAA, Web). These parameters include the type of data, the time interval, the frequency, and the format in which you want the data to be displayed (NOAA). For the purposes of my application, I needed the water temperature and tide data from the NOAA as these were the factors affecting the fish's behavior. The time range for which I needed the data exceeded the thirty day limit imposed by the NOAA's API. So I partitioned the data retrieval to thirty day intervals. The data was being retrieved via an AJAX request from the JavaScript and stored in a temporary variable that vanished after the web browser's tab was closed. This posed a problem since the data requests would take a long amount of

time due to the partitioning. So the decision was made to load all the necessary data into the application's database. Tables for the water temperature and level were made, and then indexed by time in order to speed up their retrieval. After the tables were constructed, the data from the NOAA's API was inserted into them. Using thirty day intervals starting from the first ever detection up to the last one, requests were made to the NOAA API to fetch the water temperatures and water levels and insert them to the database. Once the database was populated, it was faster to perform AJAX calls via the JavaScript.

The third variable, salinity, was attained through the USGS's (United States Geological Survey) website. Their API has similar restrictions to those of NOAA (USGS, Web). So a similar set of steps were taken to retrieve the data. A table was made for the salinity with the following columns: salinity, and time. Using thirty day intervals again from the first detection to the last one, requests were made to USGS API to fetch the water salinity and insert them into the database. Once all the data was retrieved, it was available for me to load it into my application.

Database Description

The final schema employs ten tables that are essential to the operation of the website. Figure 4 summarizes them in an ER diagram that shows the one-to-one and one-to-many relationships that exist among the different entities.

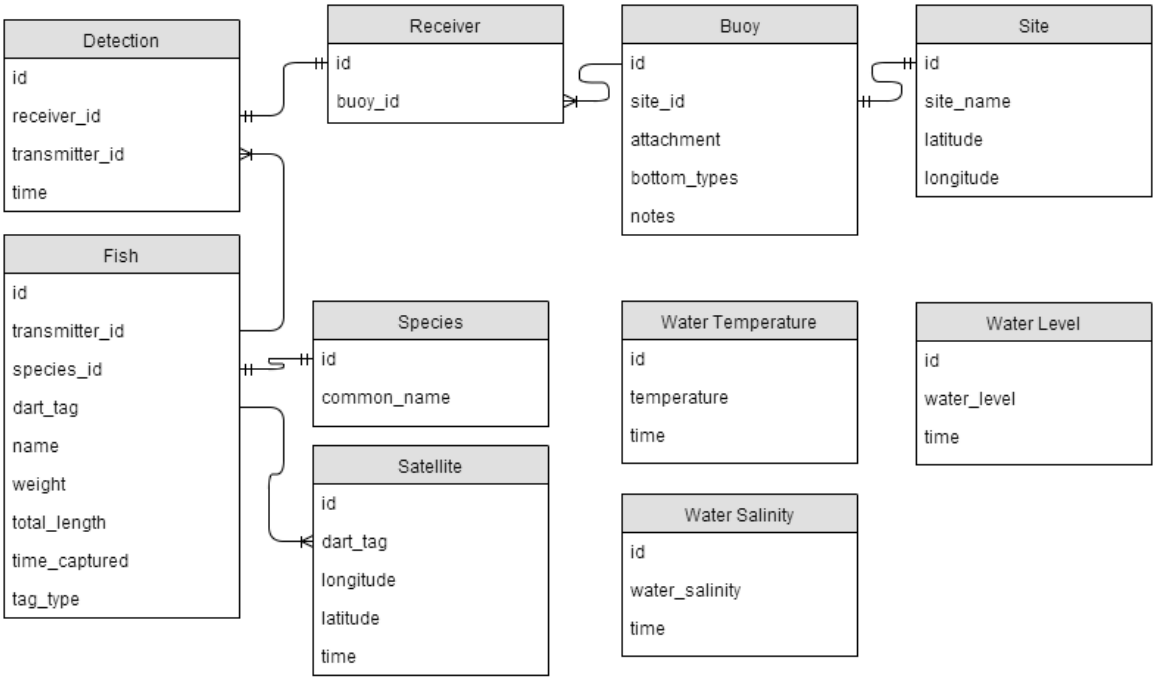

Figure 4: Database ER Diagram.

Stages of Evolution

Necessity is the Mother of Inventions

Initially, I was not a part of this project. Another student had begun working on it. I will be discussing the student's work in this section.

When the application was first developed, it was not made to keep track of fish but rather the receivers. Whenever, the biologists went to collect data by visiting the receiver locations, they would write down notes about the condition of the receiver. However, this was a difficult task due to the presence of water which would smudge their notes and also the movement of the boat made it difficult to write. After the launch of this application, they were able to keep track of everything using waterproof tablets from which they could update the receiver's status. The receivers were displayed in a table that allowed the user to modify the receivers according to its condition. Fig 5 shows what the table looked like.

Receiver no	Missing	Active	Receiver type	Buoy			
1		true		186	Show	Edit	Destroy
2		true		187	Show	Edit	Destroy
101308		true		161	Show	Edit	Destroy
101310		true		121	Show	Edit	Destroy
101314		true		172	Show	Edit	Destroy
101316		true		122	Show	Edit	Destroy
101317		true		169	Show	Edit	Destroy

Figure 5: Table Showing Receivers.

Later, functionality was added to the website that allowed the user to view static paths of the fish based on the detections. However, the data displayed was false as the detections were not ordered by time. The static paths displayed consisted of a red line that connected the arbitrarily ordered detections where the first detection was marked by a green marker and the last one was marked by a red marker. The intermediate detections were marked with small flag looking marker. The end result was somewhat chaotic as it lead to misinformation. The fish was selected of a table similar to that of the receiver and then a static track was shown. Fig. 6 illustrates what it looked like.

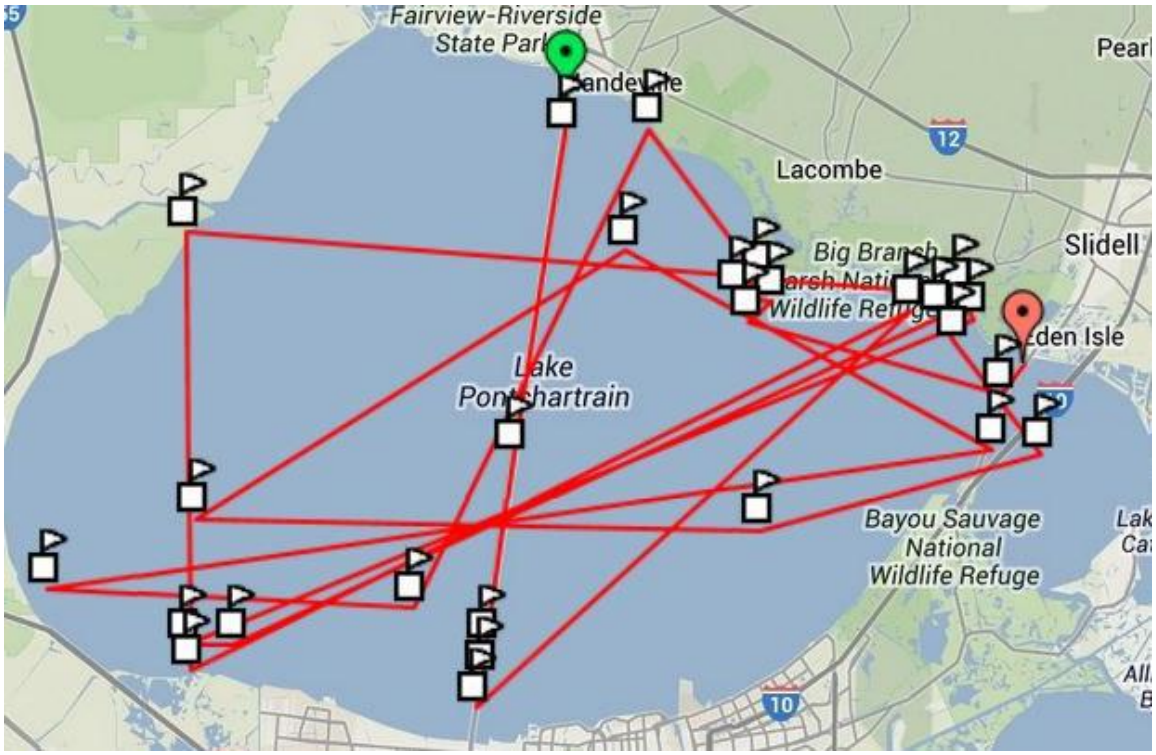

Figure 6: Inappropriately Drawn Fish Path.

The Soft Yet Essential Upgrade

After that work was done, the student, who had begun developing the application, left and then, I was assigned to take over the project. The website was approaching retirement as it did not serve any purpose to the biologists other than keeping track of the receivers' statuses. However, that task also seemed unfulfilled. The biologists using the application were unable to easily use the application on the boat to update the receivers' statuses via their waterproof tablets. So the website's focus was shifted from maintaining the receivers' statuses to tracking the fish. So instead of having the fish tracks being shown falsely with a static track, the detections were ordered by time and shown animated on the map. In addition, the user was now able to select more than one fish at a time to be displayed along with their IDs. The table in figure 7 was used to select the fish.

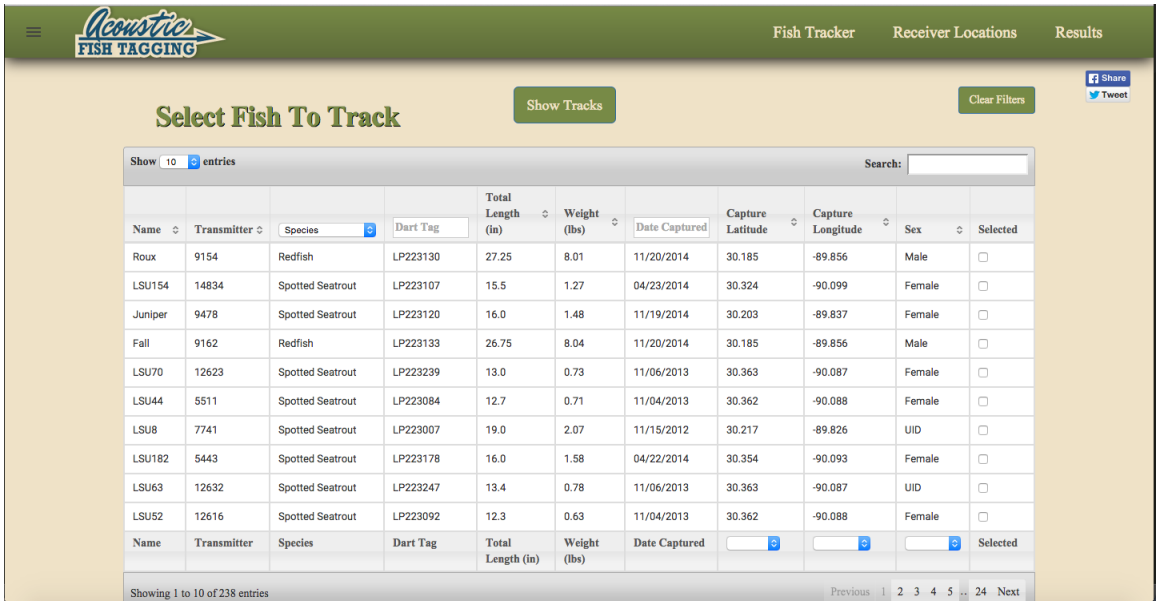

Figure 7: Fish Table to select multiple fish for tracking.

After selecting the fish to be tracked, the page would redirect to another page which displayed the fish as a small marker that moved around on the map. The marker is clickable to display the same information that was available on the table from which it was selected. A small label at the bottom left corner of the map would indicate the simulated time at which the fish was at the location. A set of buttons were also added to the top right corner in order to pause or restart the fish path animation and also to remove or draw the fish's path. Another tool that gave the user more control to the animation was a small slider above the map that allowed the user to control the speed at which the animation occurred. Other features such as recording the animation and selecting the time interval during which the detections occur were also added above the map. The end result is displayed in figure 8.

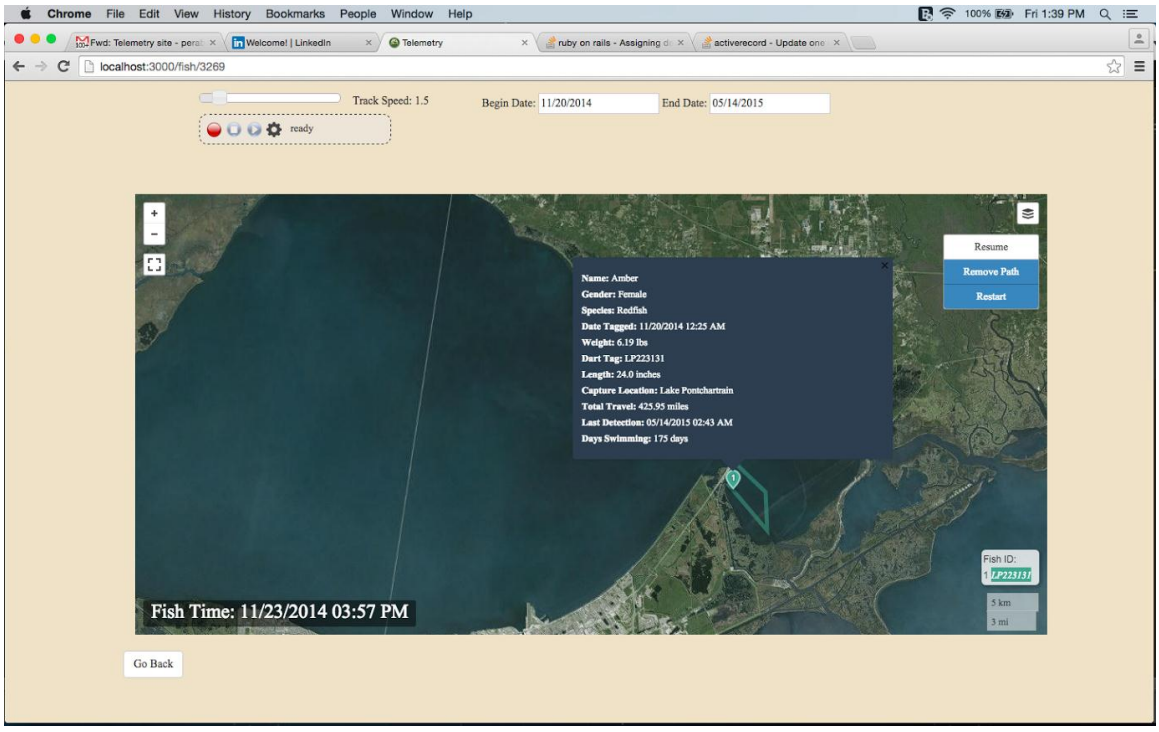

Figure 8: Map depicting clickable marker with animated fish track.

The website was also aesthetically improved to match the color scheme of *http://www.fishla.org/* as the website was going to be linked from that website. Another visual improvement was performed to the home page, which instead of being a blank page as it initially was, was then transformed to include a small table including the total number of fish tagged per species along with a small video instructing users on how to navigate/use the website. Furthermore, two small buttons were added to the top right corner under the navigation bar in order to encourage users to share the website on their social media. This is what the new home page looked like after the website was rescued:

Figure 9: The Aesthetically improved Home Page.

The Complete Overhaul

Once the website was on its feet and away from the dangers of being scrapped, a complete overhaul was performed to serve its new purpose of tracking the fish. Instead of it loading with pages like a normal website, it would load to a full map. On the right side of the map, a filter box is loaded. Inside that filter box, one can filter using a wide array

of filtering options which include: the tagging location, species, genders, season tagged, fish name, the begin date and the end date. In addition to these filtering options, the filter box also includes multiple checkboxes, each with its own added functionality. The checkboxes include a display of detections in the past four months, an option to show the water level, water temperature and water salinity along with the animated fish tracks. These features allow biologists and fishermen to compare the fish's behavior to environmental variables. These variables are displayed as labels at the bottom left corner of the map. In order to make them more appealing visually, the salinity label has a small graph which was added to it, and the temperature label has a small square that changes colors according to the temperature. Moreover, the animation controls were changed from buttons with labels to buttons with icons that are more elegant to display. The set of controls include: pause/play, draw/remove path, speed up, slow down and share. One last feature added to ornament the application was a small moon that was added in between the labels and the animation controls. This moon depicts the current status of the actual moon in the lunar cycle with respect to the simulated time on the map. Not only that, but a small moon emoji was added to the website's title. This emoji changes along with the small moon displayed at the bottom of the page. Here are some graphics demonstrating what the website looks like when it is first opened, after a fish is selected to be tracked, and when the share button is clicked:

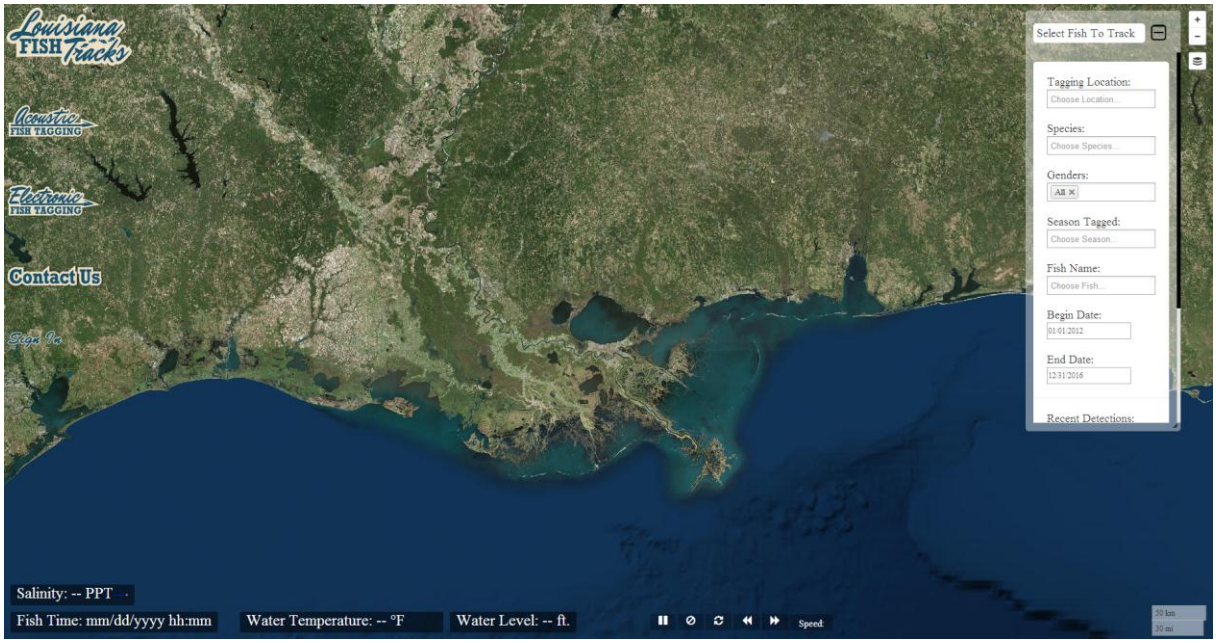

Figure 10: The new version of the website upon loading the page.

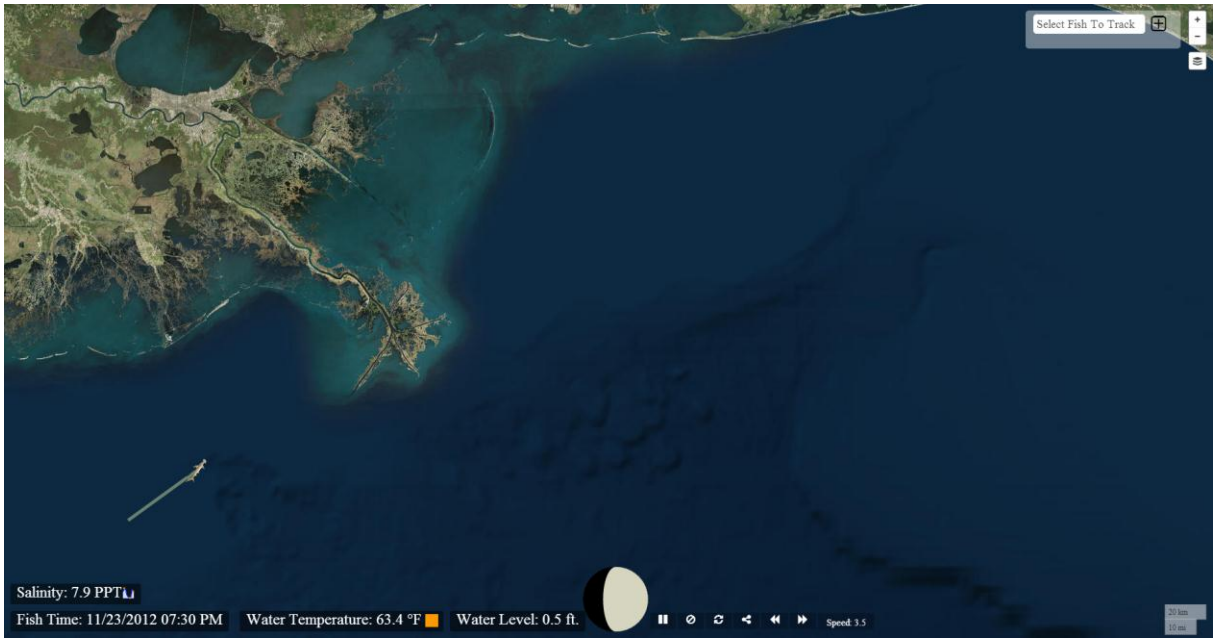

Figure 11: The application after selecting a fish to track.

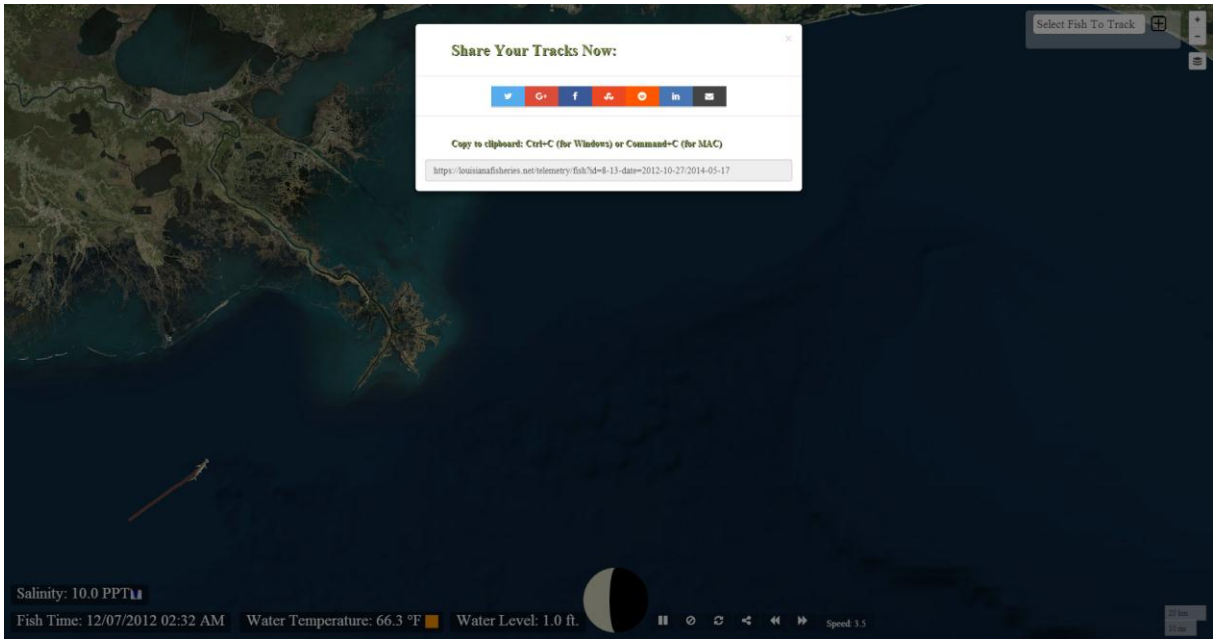

Figure 12: The application's upgraded social media sharing mechanism.

Software Complications

The Map and its Components

The first piece of the puzzle is the map. Initially the map component was displayed using Google's mapping service as it was easy to setup and implement. However, later on, the need to modify the components of required me to move away from Google's mapping service and use Leaflet maps with the Esri tiled layer service (Jgravois, Web). Esri allows us to use ArcGIS services with Leaflet, which allows us to load different kinds of maps onto the base (Jgravois). In addition, Leaflet also allows for further customization due to its compatibility with the Mapbox API (Mapbox, Web). The Mapbox API allowed the utilization of its custom features that were built for the Leaflet maps (Mapbox). This includes customized markers and controls. Two map layers were loaded onto the map and an additional layer that consists of map markings was also loaded using ArcGIS layers via Esri. This gives the user the liberty to choose what he/she wants to view when it comes to the bottom layer. Moreover, the initialization of the map is specified along with the coordinates and the zoom level of the area at which the map should be at. Ashley Ferguson, the LDWF biologist that was providing feedback for the project, asked me to initialize it with Lake Pontchartrain in the middle at a zoom level where it is visible along with the Gulf of Mexico's shoreline (Ferguson, Email). Here is a small snippet of what loading the two different base map layers along with the markings and the correct coordinates and zoom level onto the map looks like:

```
var latitude = 30.185793;
var longitude = -90.099907;

var myLatlng = new L.LatLng(latitude, longitude);

L.mapbox.accessToken = '<API_KEY>'

var worldMap = L.esri.tiledMapLayer("https://ser-
vices.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer", {
  detectRetina: true,
  attribution: "Sources: Esri, DeLorme, GEBCO, NOAA NGDC, and other contributors"
});

var oceanMap = L.esri.tiledMapLayer("https://ser-
vices.arcgisonline.com/arcgis/rest/services/Ocean/World_Ocean_Base/MapServer", {
  detectRetina: true,
  attribution: "Sources: Esri, DigitalGlobe, Earthstar Geographics, CNES/Airbus DS,
GeoEye, USDA FSA, USGS, Getmapping, Aerogrid, IGN, IGP, swisstopo, and the GIS User
Community"
});

var mapMarkings = L.esri.tiledMapLayer("https://ser-
vices.arcgisonline.com/arcgis/rest/services/Ocean/World_Ocean_Reference/MapServer", {
  detectRetina: true,
  attribution: "Sources: Esri, GEBCO, NOAA, National Geographic, DeLorme, HERE,
Geonames.org, and other contributors"
});

var myOptions = {
  center: myLatlng,
  zoom: 10,
  layers: [worldMap],
  attributionControl: false
};

map = new L.Map('map_canvas', myOptions);

var baseMaps = {
  "World Map": worldMap,
  "Ocean Map": oceanMap
};
```

Figure 13: Snippet of code depicting the initialization of a Leaflet map using ArcGIS tiles via the Esri API.

The Filter Box

The next piece of the puzzle was loading the filter box along with all of its customizable filter options. The filter box itself is nothing special as it consists of normal DOM elements, mainly divs. What required some amount of work is getting it to minimize and maximize without the whole thing disappearing. For this purpose, the box was structured

in a specific manner. The outer shell of the filter box consisted of a div that was semi-transparent. This served a dual purpose. The first is that we will be able to distinguish it from the other parts of the box and the second is that it makes it aesthetically better. As for the inner elements, there were only two, both of which were also divs with customized CSS styling. The top div consisted of the title and an icon for minimizing and maximizing the filter box. The minimizing and maximizing feature was easy to implement since I used jQuery's slideToggle() function in order to specify what element I wanted this effect to be executed on (jQuery, Web). In my case, the outer box was minimized to the size of the top div and the bottom div was completely minimized. The reason why the outer box wasn't fully minimized is because if it did, then there would be nothing to click on in order to maximize it back to its original size. The top div's icon would trigger the minimizing and maximizing of the divs and the icon would change accordingly from a minus to a plus sign and vice versa. The bottom div was responsible for holding all the filtering. The filtering boxes were regular multiple select elements, which were then modified using an external library called chosen. The chosen library transforms the multiple select items into searchable boxes with selectable items in a list (Chosen, Web). This saves space and makes it more appealing to the user. The next step would be the filtering, but before we progress to that, we first need to populate our multiple select boxes. This was done via AJAX requests to the application itself as the data required was in its database. After doing so, the heaviest part yet in terms of programming came in, which was the filtering. If we were filtering for one thing only, then it would be rather simple to pick the proper fish, but since we have so many intertwined filtering options, the task is much more complicated. The box being filtered is

the fish name box as that is the selection upon which the showing of the tracks relies on. Take a look at figure 10, which was previously shown in order to get a better idea. The elements of fish name have metadata attached to them and this metadata makes the filtering possible. The metadata consists of the following attributes: tagging location, species, gender, and date tagged. These four attributes are more than sufficient to make comparisons in order to determine the correct fish to display. Whenever a criteria is picked from one of the filter boxes, an event is triggered leading to the filtering process. Arrays are initialized for each one of the filtering attributes and they are filled accordingly from the selections made by the user. After having filled these arrays, we iterate through the fish names and eliminate the ones that do not contain an element in the selected attributes. Although, this seems like a simple and straightforward strategy, a difficulty arises when no filtering option is selected for one of the attributes as its corresponding array will be empty. Since the corresponding array would be empty that would mean all of the fish have none of the attributes. Due to this, multiple cases have to be taken into consideration where some don't have any empty arrays, some have at least one of the arrays empty and some that have a combination of arrays that are empty. Due to this one simple hurdle, the code base grew from one for loop to multiple ones depending on what case it is. The code below demonstrates my point (variable names have been changed to letters so code would fit):

```
if ($("#names option").hide(), $("#darts option").hide(), o.length > 0 && n.length > 0
&& s.length > 0)
  for (var i = 0; i < s.length; i++)
    for (var u = 0; u < o.length; u++)
      for (var d = 0; d < n.length; d++) c += i == s.length - 1 && u == o.length - 1
&& d == n.length - 1 ? '[species="' + s[i] + '"][gender="' + o[u] + '"][location="' +
n[i] + '"]' : '[species="' + s[i] + '"][gender="' + o[u] + '"][location="' + n[i] +
'"],';
else if (o.length > 0 && s.length > 0 && 0 == n.length)
  for (var i = 0; i < s.length; i++)
    for (var u = 0; u < o.length; u++) c += i == s.length - 1 && u == o.length - 1 ?
'[species="' + s[i] + '"][gender="' + o[u] + '"]' : '[species="' + s[i] + '"][gen-
der="' + o[u] + '"],';
else if (o.length > 0 && 0 == s.length && n.length > 0)
  for (var i = 0; i < o.length; i++)
    for (var u = 0; u < n.length; u++) c += i == o.length - 1 && u == n.length - 1 ?
'[gender="' + o[i] + '"][location="' + n[u] + '"]' : '[gender="' + o[i] + '"][loca-
tion="' + n[u] + '"],';
else if (0 == o.length && s.length > 0 && n.length > 0)
  for (var i = 0; i < s.length; i++)
    for (var u = 0; u < n.length; u++) c += i == s.length - 1 && u == n.length - 1 ?
'[species="' + s[i] + '"][location="' + n[u] + '"]' : '[species="' + s[i] + '"][loca-
tion="' + n[u] + '"],';
else if (s.length > 0 && 0 == o.length && 0 == n.length)
  for (var i = 0; i < s.length; i++) c += i != s.length - 1 ? '[species="' + s[i] +
'"],' : '[species="' + s[i] + '"]';
else if (0 == s.length && o.length > 0 && 0 == n.length)
  for (var u = 0; u < o.length; u++) c += u != o.length - 1 ? '[gender="' + o[u] +
'"],' : '[gender="' + o[u] + '"]';
```

Figure 14: Snippet of code demonstrating code complexity due to empty arrays.

The Movement of the Fish

The last part that completes the puzzle is the movement of the fish. The process begins after the show track button is clicked. The filtered and selected fish are then passed on to a function in the form of an array of IDs that are then used to perform an AJAX request in order to fetch their corresponding detections in chronological order. After these detections are loaded, the times from the first and last detections are stored into variables. After doing so a FishMover object is initialized for every fish that has a set of detections. Each FishMover object contains information about the detections, the derived information from it such as swimming distance and days travelled, and the fish's

corresponding marker on the map from which we can determine the fish's current location. After the initialization of all the FishMover objects, AJAX requests are performed to fetch the water temperature, level and salinity data if the booleans corresponding to each one of them are set. This data is fetched from the first time to the last time using the variables in which these times were previously stored. In a similar manner as the FishMover objects, a TemperatureData, a TideMaker and a SalinityData object are initialized. One last thing that requires initialization is the variable that keeps track of the current simulated time, it is set equal to the first time associated with the detections. After this preliminary setup, the animate function is called. This function works recursively as it calls itself repeatedly until the current time becomes greater than the time of the last detection. Once the function is called, the current time is incremented by a small fraction and calls to the FishMover, TemperatureData, TideMaker and SalinityData objects are made in order to modify some of their internal values. Each of these objects has a function call that work in the same manner. The current time is passed to the function. When the function executes, it checks to see if the time is in between the times for which data is available for. After performing this check, the difference between the two elements that have the closest time before and after the current time is stored into an element. Let's call that element Δx. Then we take the two times of the elements and calculate the percent progress using the following equation where $T_{current}$ stands for the current time of the simulation, $T_{initial}$ is the time of the first element and T_{final} is the time of the second element :

$$\%P = \frac{T_{current} - T_{initial}}{T_{\text{final}} - T_{\text{initial}}}$$

Figure 15: Equation for computing percent progress.

After calculating the percent progress, we can then determine the current value of the object. This value could be the location of the fish, or the temperature, level or salinity of the water. We determine the current value by using the following equation:

$$x_{current} = x_{old} + \%P \times (\Delta x)$$

Figure 16: Equation to calculate the current value of the element.

Here is a small code snippet showing this calculation being performed to the longitudes and latitudes of the fish detections:

```
//Compute the percent progress along this path segment.
var segmentStartTime = new Date(this.times[currentIndex]).getTime();
var segmentEndTime = new Date(this.times[currentIndex + 1]).getTime();
var pathProgress = (currentTime - segmentStartTime) / (segmentEndTime - segmentStart-
Time);

//Compute new lat/long location.
var startLat = parseFloat(this.timeSpaces[currentIndex].coordinates[0]);
var startLon = parseFloat(this.timeSpaces[currentIndex].coordinates[1]);
var endLat = parseFloat(this.timeSpaces[currentIndex + 1].coordinates[0]);
var endLon = parseFloat(this.timeSpaces[currentIndex + 1].coordinates[1]);
var curLat = startLat + pathProgress * (endLat - startLat);
var curLng = startLon + pathProgress * (endLon - startLon);
```

Figure 17: Snippet of code showing the process by which the current location is determined using the simulated time.

Conclusion

This study shows us that it is possible to track multiple animals with the help of technology. Transmitters are the first thing that will be needed. These transmitters should be attached to the animal that we want to track. Depending on the type of transmitter, the

placement of receivers is also required. They should be placed in such a manner that the acquisition of detections is maximized. Then an application/program that is capable of communicating with a database is required. A Ruby on Rails was used in the making of this project as it facilitated many aspects of the development process and it was easy to work with. The next step would be to setup the database in a similar manner as the one portrayed in figure 4. Additional tables could be added as seen necessary. Every time data is collected, it can then be loaded into the database in whatever manner the developer deems appropriate. In the case of this project, CSV files were inserted into the database since the data was readily available in that format and the insertion process was quick. The last part that we require for completing the visualization part of this application is the map component. I would recommend using Leaflet maps as they are easily customizable and have a lot of features that are provided via plugins. However, if you are more comfortable with some other service such as Google maps, feel free to do so. In the map, the developer will then utilize the data in order to display the tracks of the tracked animal correctly. This can be achieved using the equations found in figures 15 and 16.

Sources

"Acoustic Tagging." *Fish Louisiana*. Louisiana Department of Wildlife and Fisheries, 20 May 2015. Web. 11 Aug. 2015. <http://www.fishla.org/fisheries-management/fish-tagging-programs/acoustic- tagging/>.

"Bootstrap · The World's Most Popular Mobile-first and Responsive Front-end Framework." *Bootstrap · The World's Most Popular Mobile-first and Responsive Front-end Framework*. Web. 08 Apr. 2016. <http://getbootstrap.com/>.

Callihan, Jody Lynn. "SPATIAL ECOLOGY OF ADULT SPOTTED SEATROUT, CYNOSCION NEBULOSUS, IN LOUISIANA COASTAL WATERS." (2011): 1. Web. 10 Apr. 2016.

"Chosen (v1.4.2)." *Chosen: A JQuery Plugin by Harvest to Tame Unwieldy Select Boxes*. Web. 08 Mar. 2016. <https://harvesthq.github.io/chosen/>.

"CO-OPS Data Retrieval API." *CO-OPS Data Retrieval API*. National Oceanic and Atmospheric Administration. Web. 06 Apr. 2016. <http://tidesandcurrents.noaa.gov/api/>.

Ferguson, Ashley. "Telemetry Interview." Personal interview. 25 June 2015.

Ferguson, Ashley. "Transmitter and Receiver Implantation." Email Interview. 28 Mar. 2016. E-mail.

"Imagine What You Could Build If You Learned Ruby on Rails…." *Ruby on Rails*. Web. 08 Apr. 2016. <http://rubyonrails.org/>.

Jgravois. *Esri/esri-leaflet*. Github. Web. 11 Aug. 2015. <https://github.com/Esri/esri-leaflet>.

"JQuery." *JQuery*. JQuery. Web. 11 Aug. 2015. <https://jquery.com/>.

"Mapbox | Design and Publish Beautiful Maps." *Mapbox | Design and Publish Beautiful Maps*. Web. 08 Apr. 2016. <https://www.mapbox.com/>.

"PostgreSQL 9.4.7 Documentation." *PostgreSQL: Documentation: 9.4: PostgreSQL 9.4.7 Documentation*. The PostgreSQL Global Development Group. Web. 11 Aug. 2015. <http://www.postgresql.org/docs/9.4/static/>.

"Ruby-Doc.org." *Documenting the Ruby Language*. Web. 11 Aug. 2015. <http://ruby-doc.org/>.

How to Use the VR2W Family of Receivers : 44. Vemco. Web. 11 Aug. 2015. <http://vemco.com/wp-content/uploads/2014/06/vr2w-manual.pdf>.

"USGS Instantaneous Values Web Service." *USGS Instantaneous Values Web Service*. United States Geological Survey. Web. 06 Apr. 2016. <http://waterservices.usgs.gov/rest/IV-Service.html>.

Ward, Daniel. "Telemetry Meeting." Internal Documents. June 2015. E-mail.

Ward, Daniel. "Work Interview." Personal interview. 06 April 2016.