Summer Internship Report

Web server based GUI for establishing real-time communication between client side dashboard and server side AUV

Submitted By

Name: Saumya Bhatt
Placement Letter - SID1474

Under the Supervision of:

Dr. Pramod Maurya

CSIR - National Institute of Oceanography,
Dona Paula, Goa

Start Date - 3/8/2020 *End Date - 20/10/2020*

Acknowledgement

I would like to thank Dr. Pramod Maurya for providing me with this opportunity to work under his tutelage and gain valuable experience.

I would also like to express my heartfelt gratitude towards Mohit Guta, who recommended me to Pramod sir for implementing this project, for helping me out with the project and instantly clearing any doubts regarding how to proceed towards it.

Without the help and contribution of the above mentioned people, the project would not have been a success.

Abstract

The aim of this project was to build a web server based GUI, which could be able to establish a real time communication channel between the client and the server side. At the server side, an Apache engine would be running to host the necessary files which would act as the first endpoint for the AUV bot in the network.

The client side, a browser based dashboard would provide an interface to send and receive commands, display the position of the AUV and its status which included battery percent, roll, yaw, pitch, depth, etc. The latency is kept to as low as possible so as to ensure that the system works in almost real time like conditions.

The server side would host a MySQL table which would store the data from the GUI and AUV and based on requirements would be accessed either side by executing SQL queries. In this project, a simple HTML/CSS/JS based server panel has been created which would mimic the actual behaviour of the type of data the AUV would be interacting with.

Apart from receiving AUV status data, the GUI could also generate offline mapping from the coordinates received using the principles of rasterization. It also included the functionality of live streaming from the AUV using a third-party application.

Contents

Introduction

The client side GUI would be hosted on any device (example a Laptop) from where the user can control various aspects of the AUV. The server side scripts would be hosted on an Apache server of MySQL database. This would be running on a single board computer with a specialized kernel (specifically the SBC TS7550 embedded system).

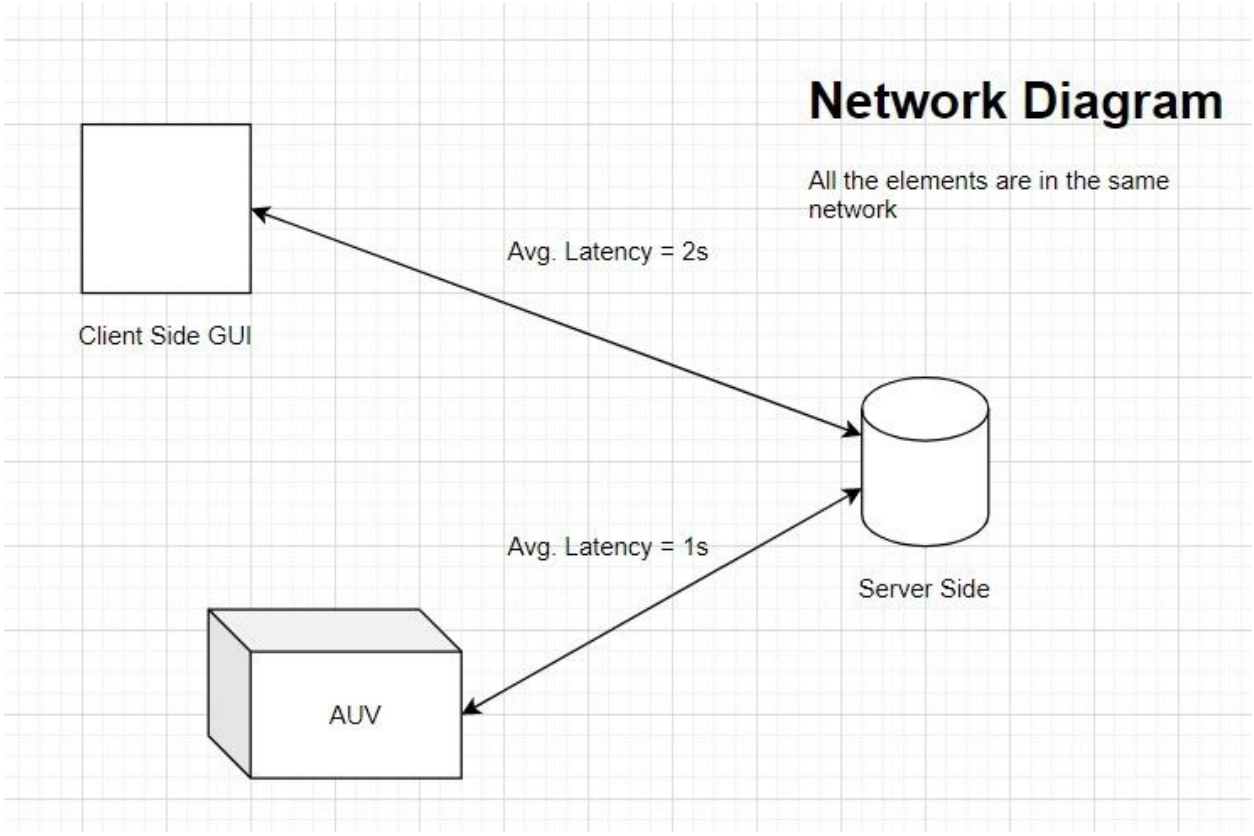

Figure 1 - Flowchart of network

Since the embedded system based single board computer is already very low on resources and memory, it was important to keep the scripts running over here very light weight. Hence the decision to use light weight languages such as JS/PHP/Ajax.

Project Outline

The work was done in versions. This was done so as to increase the complexity in a step by step method. The first 2 versions involved the use of mainly stacks such as HTML/CSS/JavaScript/PHP and Ajax to build the GUI at both the client and the server side and to update in realtime.

The latest and the 3rd version involved using Python for making the client dashboard as it provided greater control over some of the functionality and also allowed space for scaling up.

Irrespective of the version, the communication was established by querying a MySQL database using SQL queries at the client and the server side. The database itself was hosted on an Apache server.

XAMPP application was used to simulate the working environment of an Apache server and for hosting the MySQL database.

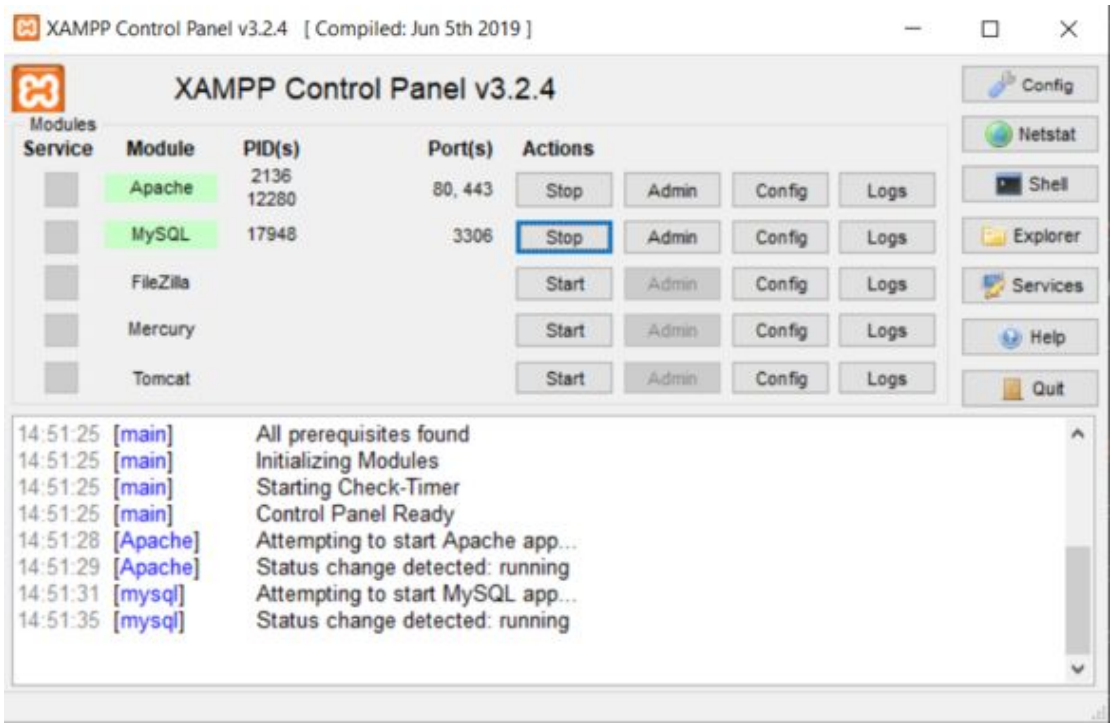

Figure 2 - XAMPP control panel

Version 1

This version involved making a HTML/CSS/JS based client side dashboard. This was very basic with the functions it would be performing not too well defined. The only functionality it had was the capability of being updated every 1s. The server side also had a similar panel to simulate the AUV. The communication was done using a MySQL database which was running on an Apache server.

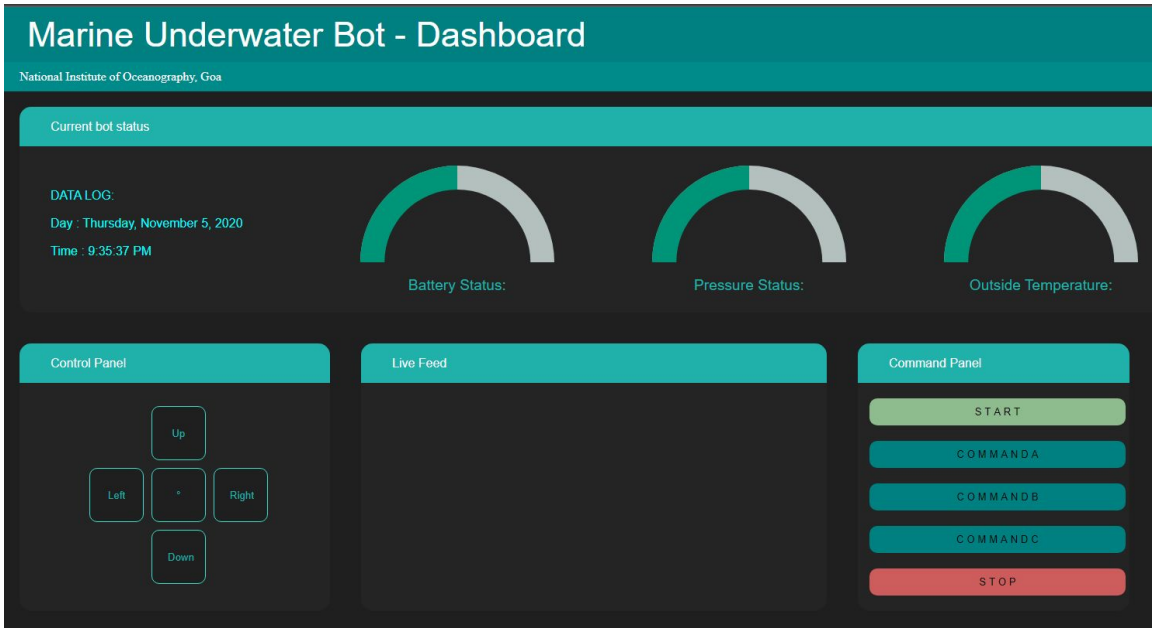

Figure 3 - Version 1, Client side dashboard

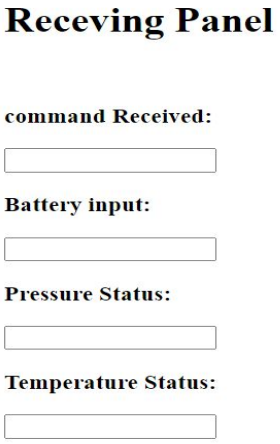

Figure 4 - Version 1, server side panel

Version 2

This version followed similar architecture as that of Version1 1. However, all the functionalities that were actually required were displayed on the GUI. However, it did not allow for mapping or live streaming video capabilities.

The most important update in this version was the inclusion of a small command prompt window within the GUI which could keep track of the mission files which were uploaded. This version was the first one which allowed for such capability.

The Latency was kept at 1s. The file upload followed the similar route of storing its contents in a MySQL database and subsequently querying them to be shown at the server side. Another important feature of this file upload was that it was able to read python files and run them and eventually store the output in the database. The running of files was done at the client side.

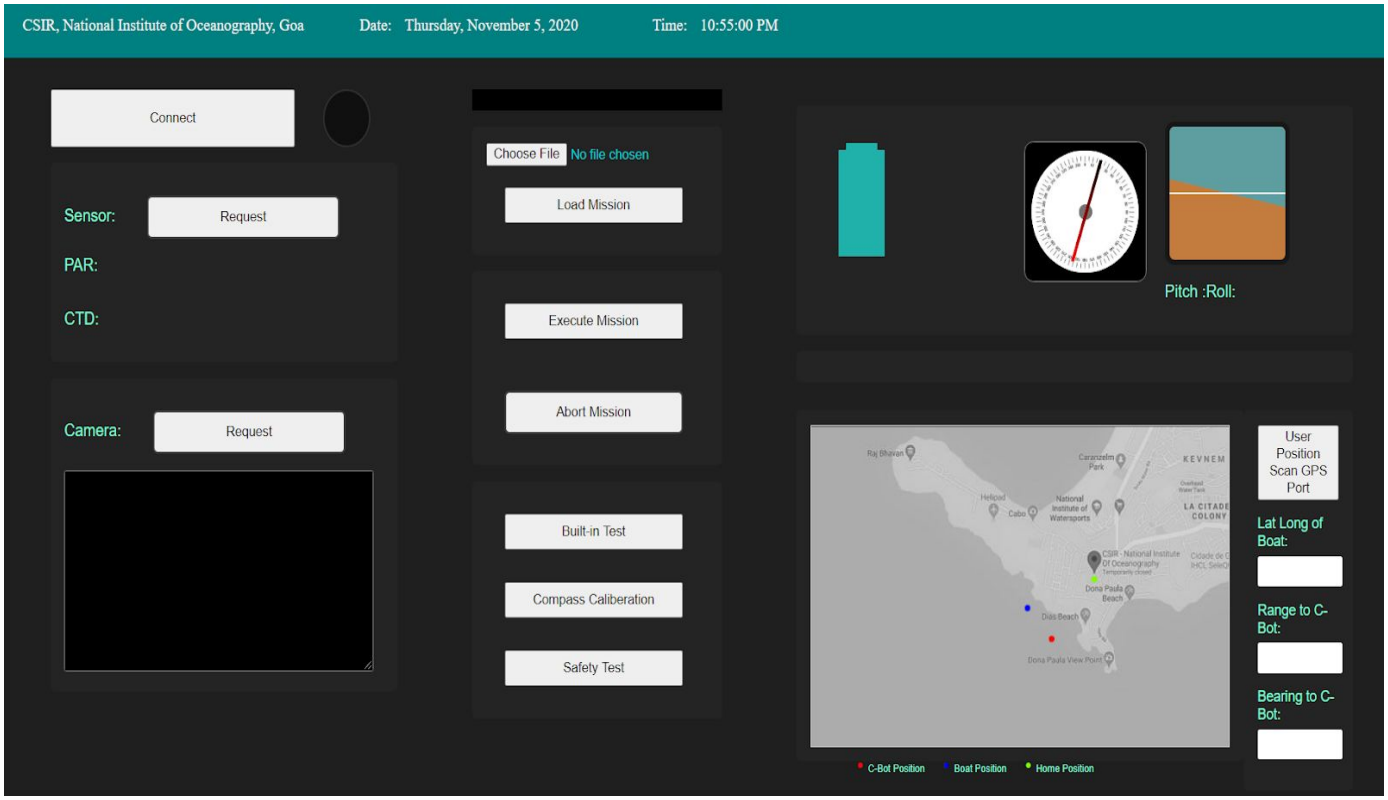

Figure 5 - Version 2, Client side GUI

One important drawback of this version was not being able to update the map dynamically. For static displaying, it rasterized an already existing image and displayed the locations (which themselves were hard coded) in points. Hence, not only could it not update the map continuously, but also if one were to display locations on another image, he/she would have to manually open the code, add the image and ensure that it followed the required number of pixels for rasterizing. This thus proved to be very cumbersome.

Shortcomings

Using regular HTML/CSS/JS for the GUI, it proved to be a very difficult task for implementing the functionalities that would actually make the dashboard useful. The following points proved to be quite difficult to do so in this versions of the GUI:

1. Dynamically updated offline mapping feature.
2. Live streaming from the AUV camera onto the dashboard.
3. Upscaling the dashboard to accommodate more features was not very easy and hence too cumbersome.
4. Provided little to no scope of having analytics features into the dashboard in the future.

It was because of the above mentioned shortcomings that the GUI was decided to be implemented in python (the client side). It subsequently was successful in answering many of the questions asked above.

Version 3

The client side interface which consisted of the GUI was made entirely in python. The Streamlit library (version 0.67.0) was used for the interface. It had the functionality of dynamically updating the AUV status values such as battery, pitch, yaw, roll, depth, etc.

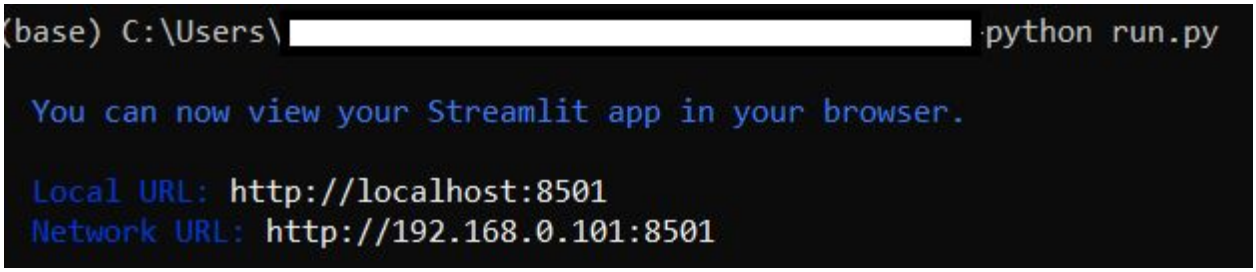

Figure 6 - Instance of the client side GUI running

Apart from this, it also has the functionality of displaying offline and online mapping which could be updated in real time via the server panel. The communication in both of the above cases was done using a MySQL database running on an Apache server.

It could also handle file traffic, as one of its functions was to read and upload the contents of the mission file uploaded at the client side and send it to the server for execution. Apart from this, it used a third party application (IP Webcam) to simulate the working of live streaming from the AUV.

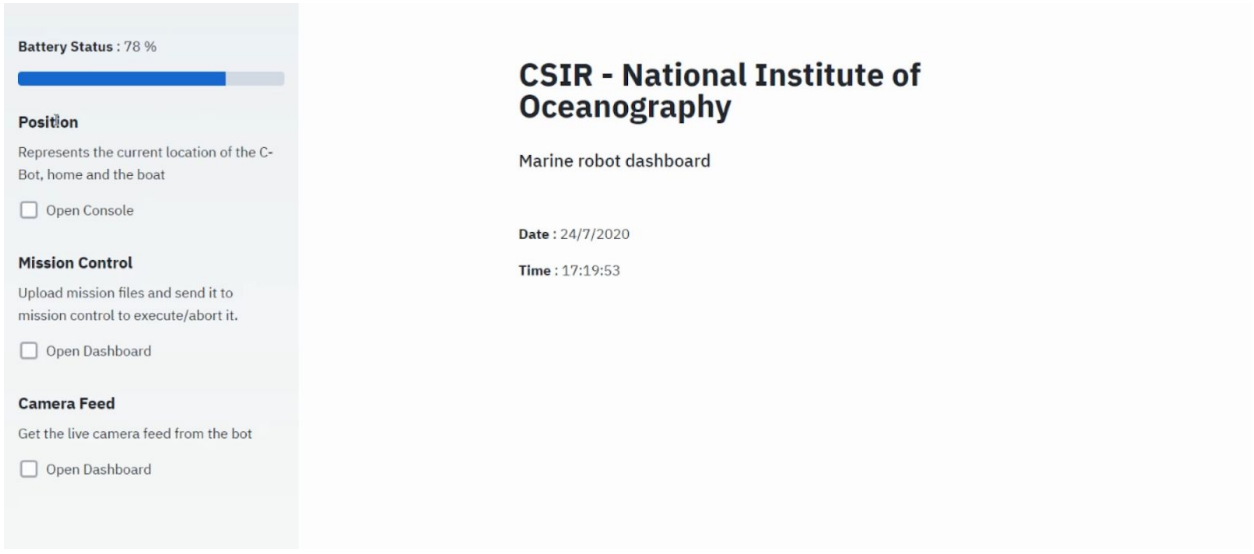

Figure 7 - Screenshot of the GUI

Mapping

The locations that the GUI would get were currently updated manually by the server side panel. These locations would get stored in the MySQL database and were extracted by executing SQL queries at an interval of 2s. This time limit was specified so as to ensure that the dashboard (which runs synchronously and thus uses another python library - schedule, so as to run the real time functions in the background) do not hang or generate lag.

Offline Mapping

The offline map takes in 2 files before displaying the actual map. First is the image file. Since mapping here is done by rasterizing an image and plotting scatter plots on top of it using the matplotlib library, it required a background image which could act as a reference for the map. If another location's offline mapping is required, that location's map image could be uploaded as well thus making this feature generalized.

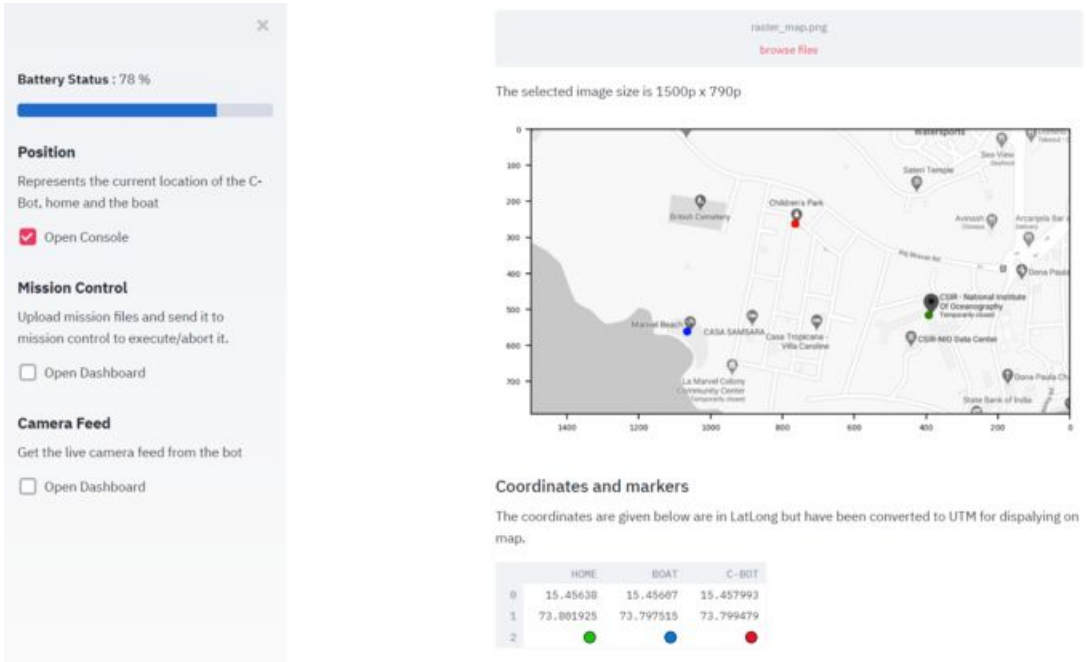

Figure 8 - Screenshot of the offline image widget

Another input which the offline map required was a refloc.txt (reference location) file. This file would contain the actual coordinates, pertaining to the top left and the bottom right corner of the image file which was previously uploaded. This could be done in advance and thus would not interfere with the dashboard's working.

This process of uploading an image and a reference file is to be done only once while using the locations of that area, as the locations would get updated automatically.

The reference locations in the refloc file would first be converted from lat long to UTM so that it could be applied on cartesian coordinates. Later, using it and image dimensions of the uploaded image could be computed so as to rescale the locations such that they fit into the map widget along with the uploaded background mapped image. The rescalling adjusted for an accuracy of 1-2 meters.

Online Mapping

The online map does not require any input from the user side. On opening the widget, the map automatically starts showing the map in 3d format. This was achieved using the python library PyDeck. Note that, Online map requires access to the internet only at the Client side and works imperative of the presence or absence of internet at the server side.

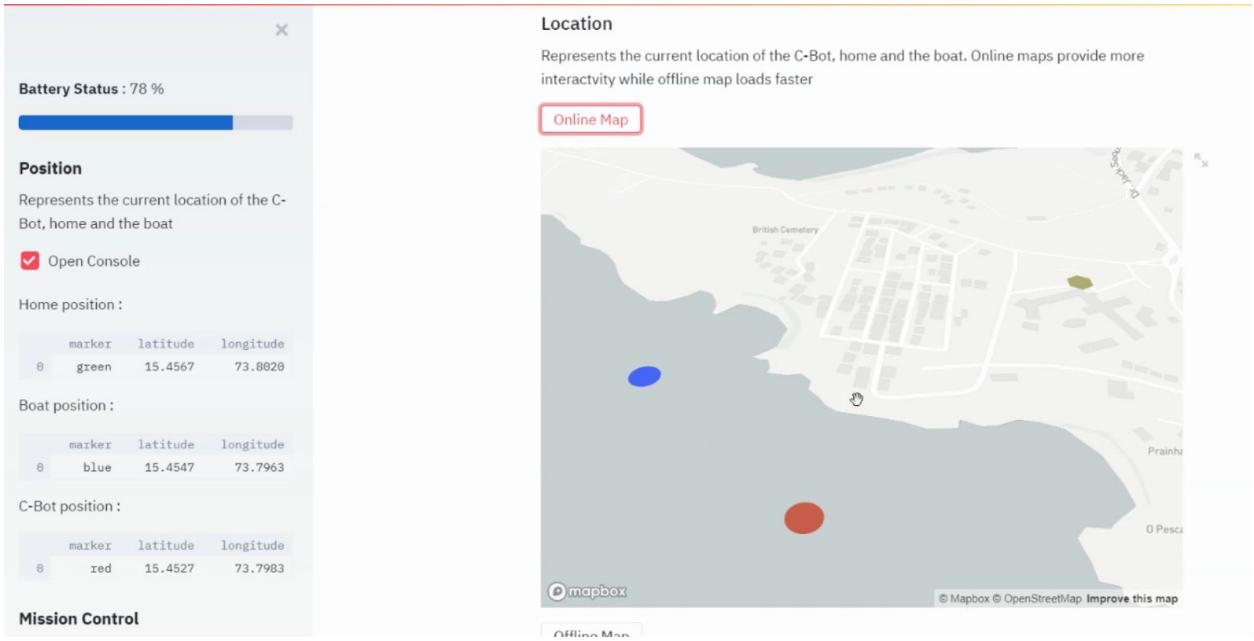

Figure 9 - Screenshot of online mapping feature

The major advantage of using an online map over offline is that the map widgets could be dragged around to display other locations. It could also be zoomed in or out thus giving greater usability. Offline map only points out the locations onto the map but does not give any such features of dragging around or zooming.

File Uploading

The AUV will receive instructions regarding its future coordinates, tests to run, etc via a text file. Transferring the contents of that text file from the client to the server was done using the MySQL database. Everything was done in real time with a latency of 1s.

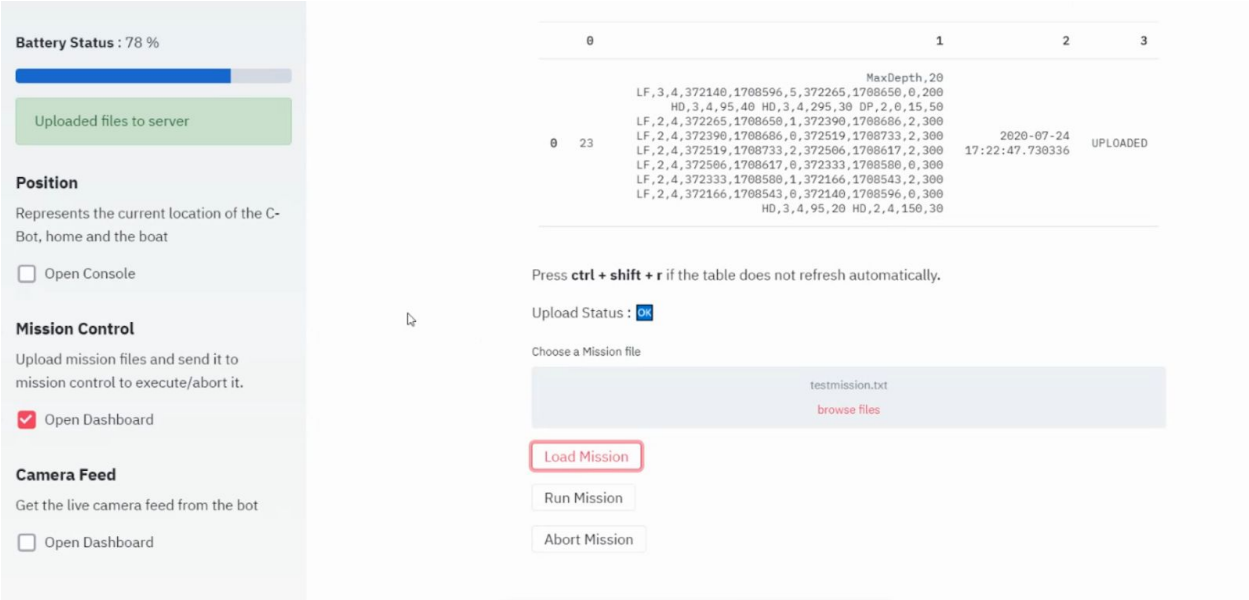

Figure 10 - Screenshot of file uploading feature

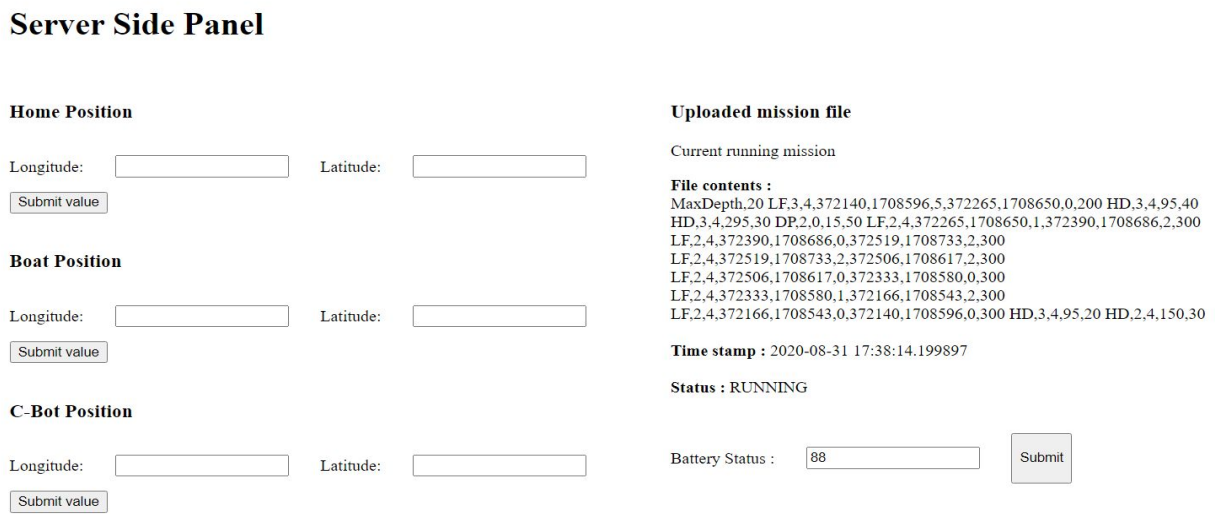

Figure 11 - Server side panel for simulation of AUV

Live Streaming

This functionality involved installing the IP webcam application on the AUV or any other platform from where we want to live stream. On starting that application, it would generate an IPv4 address. Ensuring that the GUI and the platform on which the IP webcam application has been installed, entering the generated address into the console at the dashboard, a new window would pop up, displaying the live streaming.

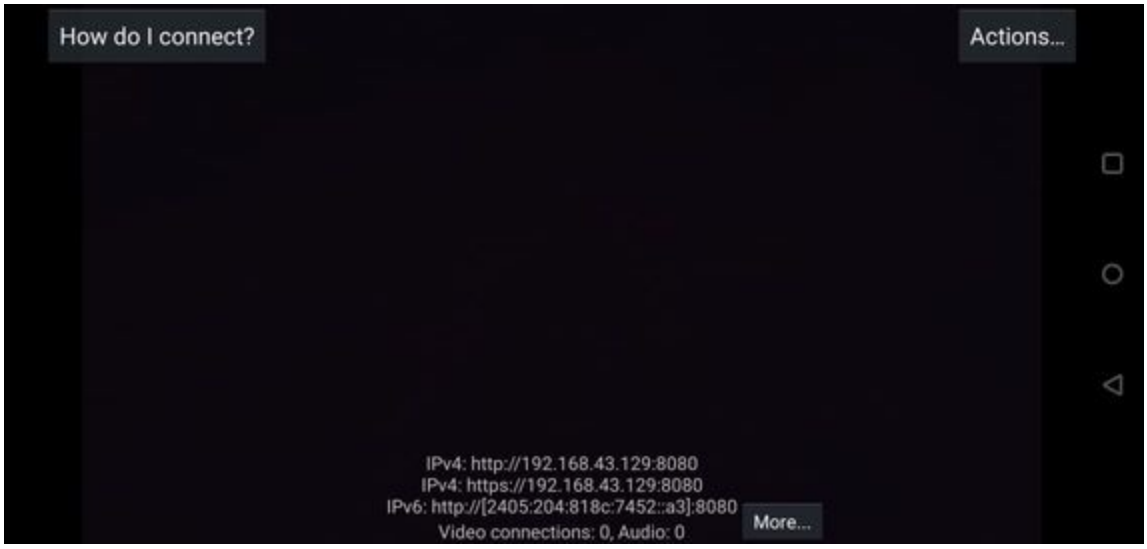

Figure 12 - IP address generated at the IP webcam application.

The dashboard receives each frame which is compiled at a spacing of 0.5s so as to appear as a moving picture. This was achieved by running a python script using the OpenCV library.

Database Structure

Since, this version of the dashboard allowed control of just one AUV, there is just one database which consisted of multiple tables, each one serving as the communication node for one parameter of the AUV's status. In the future work when multiple AUVs would be connected to the GUI, instances of the Database would be created on the go when an AUV would be assigned a unique ID.

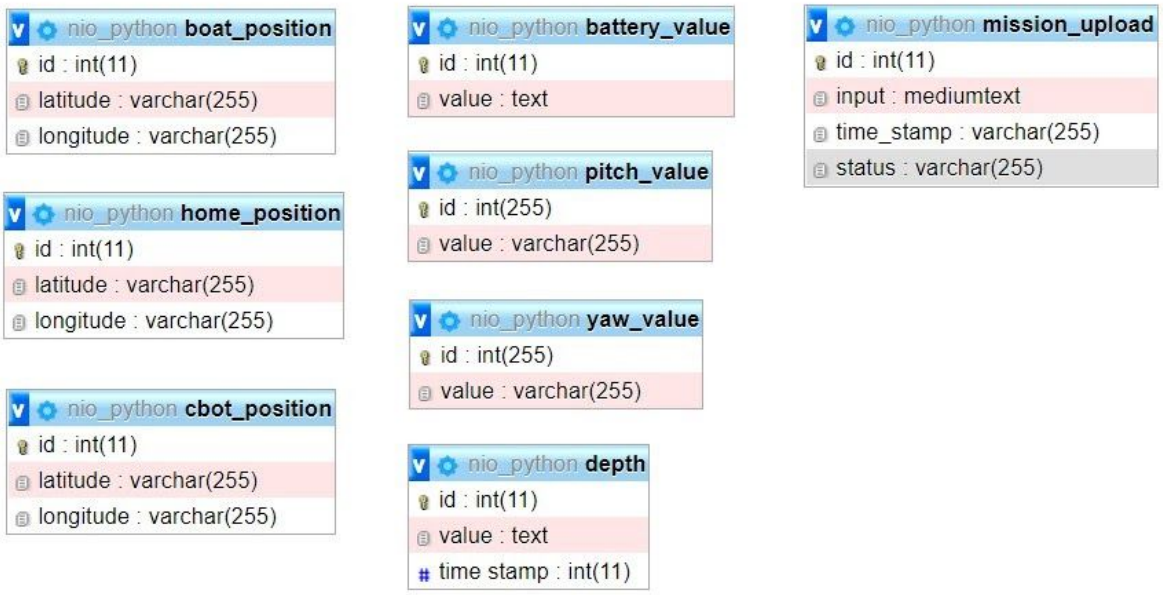

Figure 13 - Tables within the database

In the future work with multiple AUVs, the tables within each database instance would be joined to form a simplified structure and thus reduce boilerplate code when querying from each table for a data.

Advantages of version 3

Having a python based GUI at the client side has numerous advantages over having a traditional dashboard made using HTML/CSS/JS/PHP. These advantages are mentioned below:

1. **Integrating complex functionalities within the dashboard.** Since the backend is written in python, many functions such as mapping, complex computations, etc could be integrated with more ease.

2. **Better ease of upscaling**. The Streamlit library takes advantage of having its own in-built HTML/CSS/JS code so that there's minimal fuss in worrying about the frontend. It has built in classes for each element of the frontend, thus the user can focus work on the backend. This invariably leads to a better and more easier method of upscaling the functionalities and UI/UX of the GUI.

3. **Adding machine learning and data analytics to the GUI.** Since it is written completely in python, there is little to no headache of writing an interface which connects the ML/DA applications to the frontend. This in turn leads to a much more useful and an interactive dashboard than the traditional ones.

4. **Smaller on-disk size**. Since it uses object-oriented methods for serving both the frontend and the backend, there is very little code to write in comparison to using HTML/CSS/JS for the frontend and PHP/Ajax for backend when working with a traditional GUI.

Future Work

The current dashboard mainly deals with just one AUV at the moment. The plan is to upscale it to handle multiple numbers of AUVs. This would help to have a single point of control for all the AUVs rather than have a unique GUI for each one which would not only prove costly, but also impractical if there are a significant number of them.

Also, the current communication channel works over the MySQL database with multiple hanging tables. When dealing with multiple AUVs, this can become a bit slow. Thus, the database schema could be updated to support this structure as well make it more streamlined to query it.

The live streaming functionality currently uses a third party application for working. This has to be optimized to work with the stack that the AUV would be carrying with itself. Integrating and deploying the server side code with the linux based single computer running on an embedded system would be done in further works.

Optimize the client and the server side code so as to keep the latency at its minimum and thus achieve almost real-time two way communication channel. This optimization would also include optimizing various other functionalities of the dashboard to better improve the performance and the user experience.

References

The following python libraries and backend stacks were used for the completion of this project:

- https://docs.streamlit.io/en/stable/
- https://pandas.pydata.org/docs/
- https://numpy.org/doc/stable/
- https://matplotlib.org/contents.html
- https://schedule.readthedocs.io/en/stable/
- https://api.jquery.com/Jquery.ajax/
- https://www.php.net/docs.php
- https://pydeck.gl/layer.html

The following third party applications were used for the completion of this project:

- https://dev.mysql.com/doc/
- https://docs.joomla.org/XAMPP
- http://ip-webcam.appspot.com/static/doc.html