

**Boston University**

**Electrical & Computer Engineering**

**EC464 Senior Design Project**

**Second Prototype Testing Report**



**Team 16:** The Sharks

Lucía Martínez Ruiz [lmrpueyo@bu.edu](mailto:lmrpueyo@bu.edu)

Robert D’Antonio [robdanto@bu.edu](mailto:robdanto@bu.edu)

Xinglin He [hxl@bu.edu](mailto:hxl@bu.edu)

Zhaowen Gu [petergu@bu.edu](mailto:petergu@bu.edu)

Lydia Jacobs-Skolik [ccjs@bu.edu](mailto:ccjs@bu.edu)

Submitted: Mar. 8th

# **1. Abstract**

Since the first prototype test, we have made considerable progress toward our goal of developing an implementation of the SLAM algorithm tailored to underwater robotics. Shortly after the start of the spring 2024 semester, we split our project into separate sub-projects. Robert and Lydia’s work primarily focused on data acquisition and generation, with Robert continuing his work with HoloOcean and Lydia working with real-world sonar data from the Basurelle Sandbanks in the English Channel. Lin and Lydia worked together on developing edge-detection technology to classify artifacts in sonar data that could be used as “landmarks” for the SLAM algorithm. Peter worked on developing the Sonar-SLAM algorithm itself. This report details distinct approaches that will be used to demonstrate the functionality of each of these sub-projects and concludes with a precise plan of how we will combine these sub-projects to produce an implementation of the SLAM algorithm tailored to underwater robotics.

# **2. Required Materials**

**2.1.** Hardware:

* Linux PC with Nvidia Tesla V100 GPU and OpenGL installed
* Generic PC (preferably Windows)

**2.2.** Software:

* Python 3.0 libraries
  + HoloOcean (and its dependencies)
  + OpenCV
  + NumPy
  + matplotlib
  + seaborn
  + Pandas
  + pyxtf
* C++ libraries
  + OpenCV: >= 3.0
  + boost
* Cmake: >= 2.8
* Visual Studio: >= 2015
* VMware workstation pro >= 17
* ImageJforXTF

# **3. Set Up**

As our project is entirely in software, most of these tests can be performed on any PC as long as all required packages and libraries are installed and accessible. The exception is the HoloOcean underwater simulator, which (from our experience) requires a Linux system with a Nvidia Tesla V100 GPU. Today, we will use a V100 GPU available on the BU SCC. For Blue-rov slam, the program is really unstable due to the publisher’s predilection in using Python instead of C++ for ROS. It may have stack errors from time to time.

# **4. Pre-Testing Setup Procedures**

**4.1.** Holoocean:

1. Connect to a desktop on BU’s SCC with a V100 GPU
2. Run the following commands to prepare HoloOcean:
   * cd /projectnb/ece601/SlamSeniorProj2023/HoloOcean\_scripts
   * source startHolo.sh
3. Run HoloOcean files using python <<filename>>
   * Primary working file: nav\_agent.py

**4.2.** Basurelle Sandbanks

1. Download ReadSonar.py from <https://github.com/peterguzw0927/Senior_Design>
2. Install the ImageJforXTF from <https://sourceforge.net/projects/imagejforxtf/>
3. Download the sandbank datasets: <https://data.europa.eu/data/datasets/processed-side-scan-sonar-data-from-bassurelle-sandbanks-sci?locale=en>

**4.3.** Edge Detection

1. Download the zip file or git clone this repository: <https://github.com/hxl1236/Edge-detection.git>
2. Extract the files from the zip folder.

**4.4.** SonarSLAM Algorithm

1. Open VMware workstation pro 17
   * Power up virtual machine
   * Download dependencies:  
     https://github.com/jake3991/sonar-SLAM/tree/main

# **5. Testing Procedure**

**5.1.** HoloOcean:

1. Run nav\_agent.py, which tests the HoloOcean side-scan sonar simulation on a vehicle with a lawnmower navigation path
2. Run log\_read.py, which reads the log file written by nav\_agent.py

**5.2.** Basurelle Sandbanks

1. Run ReadSonar.py in the terminal, which reads one of the xtf files from the sandbanks datasets.
2. Open ImageJforXTF and use it to read the same xtf file.
3. Both files will produce the same sonar images for comparison.
4. Both files also read all the information stored in each ping of the image.

**5.3.** Edge Detection

1. In the terminal:
   * cd to the path containing all these Python files with corresponding images.
   * Run python testingCurlyTorpedo.py
   * Run python testingWholeImg.py
   * Run python testingCutWhole1.py
   * Run python testingCutWhole2.py
2. Each of the codes tests on a different sonar image and produces a final filtered image, highlighting the contours of detected landmarks.

**5.4.** SLAM Algorithm

1. Open up a new terminal
   * cd ws\_sonar/
   * catkin build
   * bash s1\_run.bash
2. Open up a new terminal
   * cd ws\_sonar/
   * bash s2\_run.bash
3. Download image and save data after completion

# **6. Measurable Criteria**

**6.1.** Holoocean

1. Produces a reasonable and usable output that can be deciphered into an image using the log\_read.py utility program

**6.2.** Basurelle Sandbanks

1. The code should be able to read a .xtf file and generate a sonar image.
2. It will show you the information (such as pitch, roll, depth, x, and y coordinates) of one ping.
3. The sonar image generated by the code should match the image generated by ImageJforXTF.

**6.3.** Edge Detection

1. It should be able to filter out noises (such as debris) on the sonar images.
2. Identify large landmarks (torpedo) on sonar images with a success rate >= 70%.
3. Output the number of landmarks on sonar images.

**6.4.** SLAM Algorithm

1. Run the program multiple times and compare the final results.
2. Did the program work?
3. Is the point cloud similar?
4. Is the trajectory constant?

# 

# **7. Test Results**

**7.1.** HoloOcean

The HoloOcean simulator continues to show potential as a means of generating synthetic data. However, slow runtimes and poor documentation have been major hurdles in getting this program to work for our purposes. Currently, we are able to navigate an object through the ocean environment while collecting sonar data. However, we are not yet able to generate the data we would like. We intend to put in another week’s worth of work on this program before reaching out to the Frost Lab (HoloOcean’s development team) directly to see if they are willing to assist.

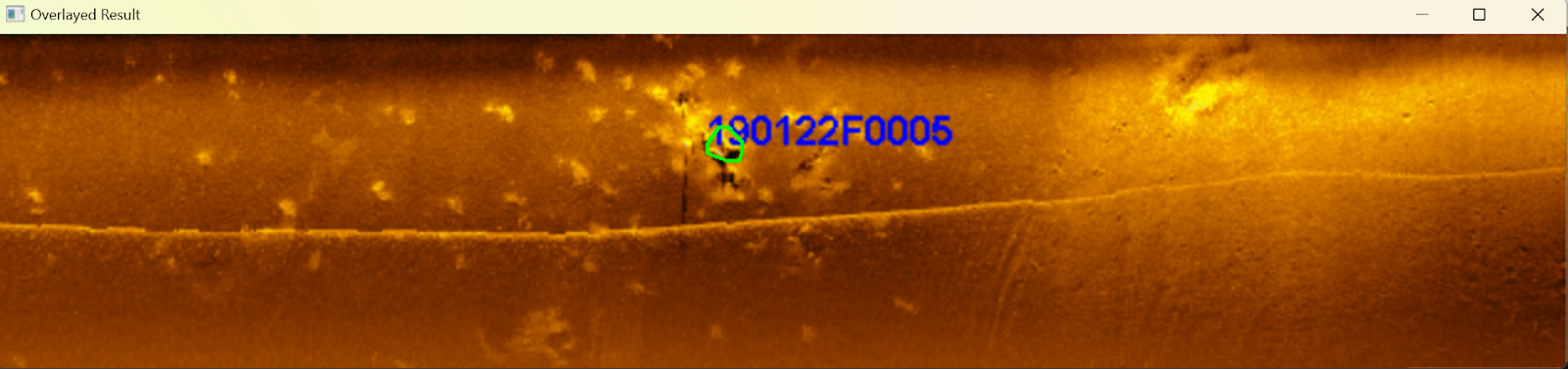
**7.2.** Basurelle Sandbanks

The algorithm successfully turns each of the ‘.xtf’ files from the sandbank dataset into an image segment. It can stack every ping on top of each other and create a sonar image showing one part of the trajectory of the vehicle.

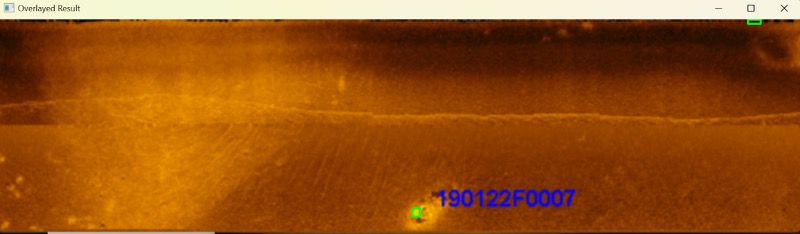
**7.3.** Edge Detection

The edge detection algorithm works in the following ways. First, it draws the contours on the darkest areas in the sonar image. Then, it replicates a similar process for the brightest region. Ultimately, it merges the closest contours from the dark and bright regions, thereby revealing clear landmarks, notably torpedoes. We tested the algorithm with four different sonar images.

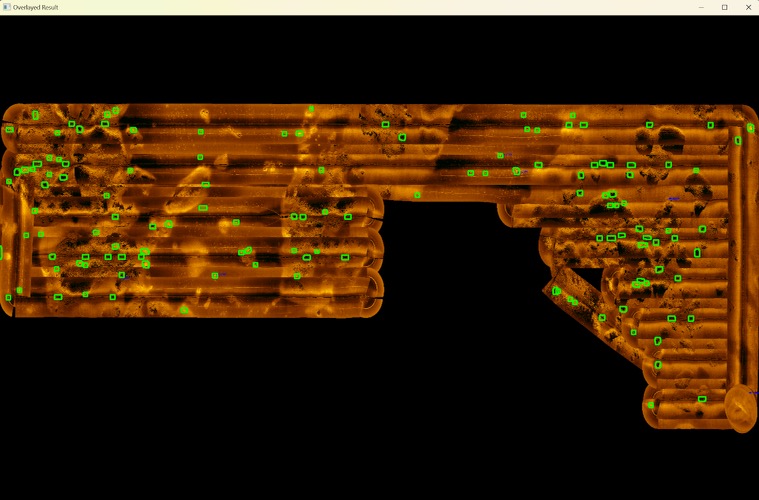
In the first image, we can see the detection of a landmark. The number label indicates that it is a founded torpedo by the human eye. We achieved a 100% success rate here.



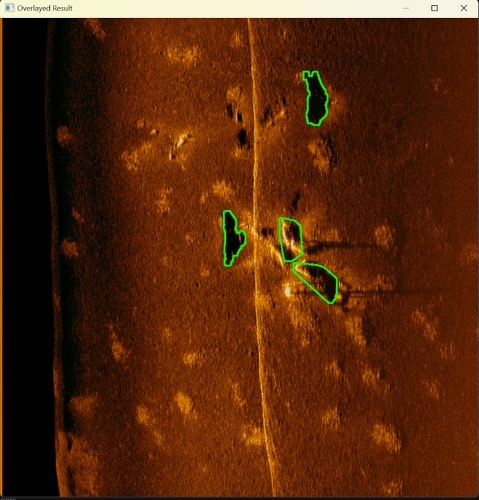
The second image shows a similar result as the first image, which is another 100% success rate.



This third image represents the entire trajectory of the vehicle. This aims to prove if the algorithm can detect all the landmarks in a bigger dimension. Although the accuracy is lower as it only detected 4/6 of the predicted landmarks labeled with numbers as in the previous images, the algorithm demonstrates the potential to work with huge paths.

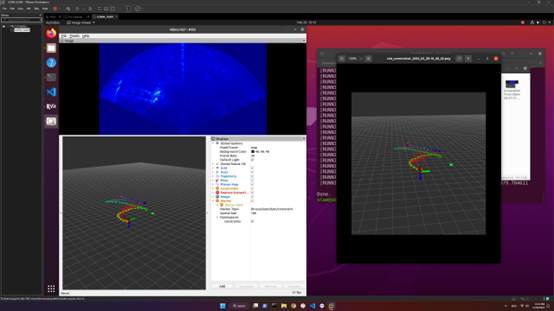


The last sonar image has a higher quality and finer details. It also achieves a 100% success rate in the detection.

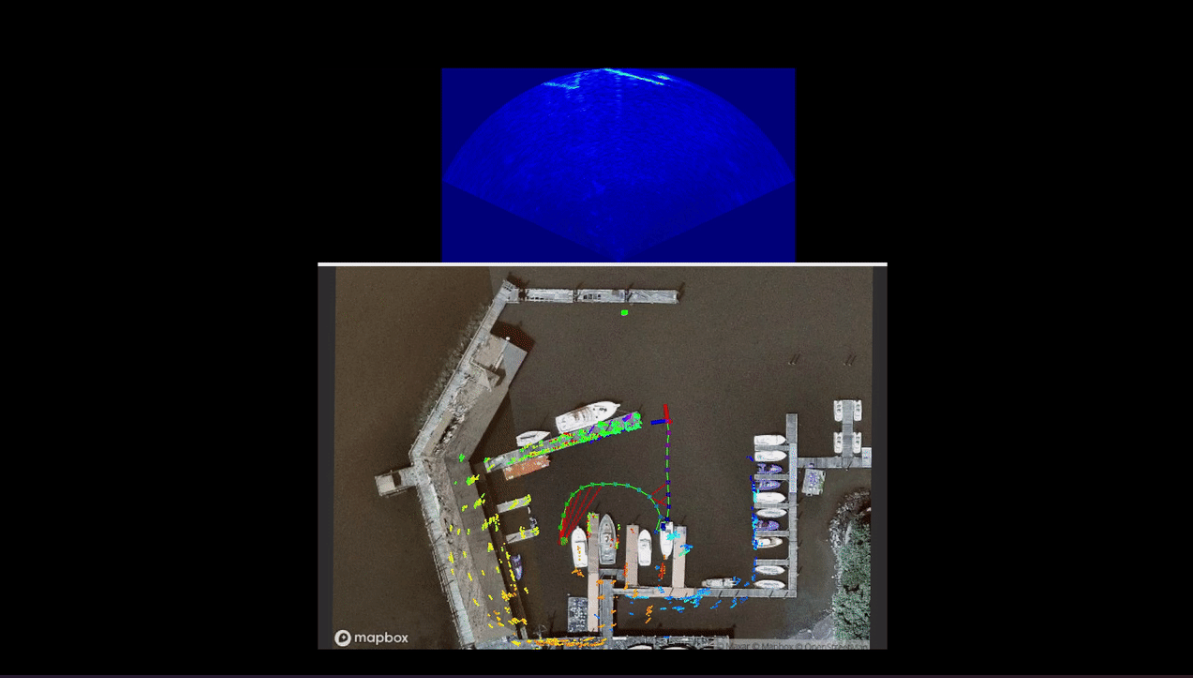


**7.4.** SLAM Algorithm

The blue shell-like interface at the top is the imaging sonar display. It showcases the data collected by our sonar system and in this case we can clearly see the landmarks. This algorithm is going to run a feature selection node on the sonar image and extract the potential landmarks. After extraction, it will compare the landmark location and the dead-reckoning information (i.e. information collected by Doppler Velocity Log, Inertial Measurement Unit) and update the vehicle's location. In the bottom left part, we can see the algorithm's estimated trajectory varying with time. The red line connecting the trajectory is where the update in information happens. If the vehicle repeatedly sees a landmark, it will update its position accordingly incorporating the dead reckoning data. By comparing the results from two consecutive runs (the right side is the output of the algorithm from the previous run), we can conclude that both procedures obtained the same point cloud landmarks and the same trajectory which means the algorithm is successful and stable.



There is the trajectory of the underwater vehicle in a bird side view. We can see the trajectory of the ground truth matches the trajectory predicted by the slam algorithm. Proofing the algorithm correctly identifies its trajectory.



# **8. Test Conclusions**

**8.1.** HoloOcean

* This round of testing showcases the limit of HoloOcean when it comes to accomplishing our goal of data acquisition. While there is still significant potential from this program, poor documentation, a lack of online examples, and long runtimes have led to very slow productivity. A reasonable next step—and one we will take—is to reach out to the Frost Lab at Brigham-Young University, which is responsible for the development and maintenance of the HoloOcean simulator. Ideally, they will be able to provide suggestions for how we can effectively and efficiently construct a simulation file to acquire the necessary data for our project.

**8.2.** Basurelle Sandbanks

* Since we are now able to read a .xtf file from the sandbank dataset and produce actual sonar images, we can use it to generate more sonar images to test the edge detection algorithm. Also, by obtaining the position information from the .xtf file, it won’t be too complicated to sort the sonar images based on the coordinates and draw the entire trajectory of the vehicle.

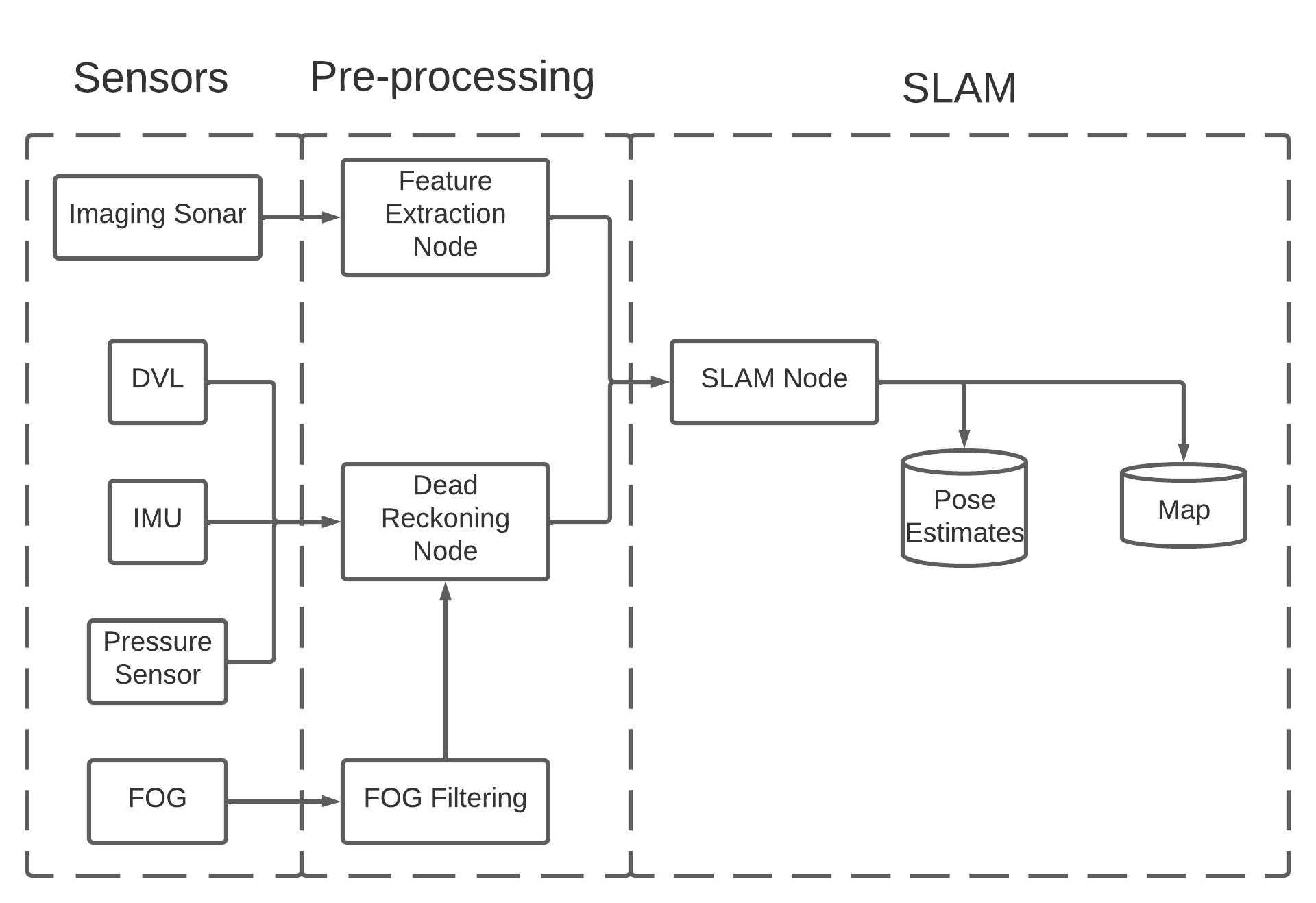
**8.3.** Edge Detection

* The current edge detection algorithm is reliable because it can successfully detect the correct landmarks on multiple sonar images. By incorporating this algorithm with the sonar images and position coordinates of the vehicle extracted from the sonar dataset, we should be able to generate a global map storing both the locations of the landmarks and the vehicle. Then, we could develop a landmark-matching algorithm that can match the shapes of similar landmarks when the vehicle revisits the same location. If the landmarks are matched, we can perform loop closure to improve the accuracy of the vehicle’s trajectory.

**8.4.** SLAM Algorithm

* The Blue-rov is viable when deploying in the virtual machine, and the effect of the slam algorithm way precedes the ORB3-SLAM since all of its data has two sources, one from the imaging sonar’s feature extraction node and the other from the dead reckoning node. The slam algorithm combines both inputs and runs through a very sophisticated loop detection algorithm, and mapping algorithm and finally comes up with a precision measure of the trajectory and landmarks. (Figure 1.)

**9. Appendix**



**Fig. 1.** SLAM overview