Boston University
Electrical & Computer Engineering
EC464 Senior Design Project

Final Testing Report

Team 16: The Sharks

Lucía Martínez Ruiz lmrpueyo@bu.edu
Robert D'Antonio robdanto@bu.edu
Xinglin He hxl@bu.edu
Zhaowen Gu petergu@bu.edu
Lydia Jacobs-Skolik ccjs@bu.edu

Submitted: 4/4/2024

1. Abstract

Since the start of the spring 2024 semester, we have made considerable progress toward our goal of developing an implementation of the SLAM algorithm tailored to underwater robotics. 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
  - scipy
- C++ libraries
  - OpenCV: >= 3.0
  - boost
- Cmake: >= 2.8
- Visual Studio: >= 2015
- VMware workstation pro >= 17

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 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: `lawnmower_hovering.py`

4.2. Basurelle Sandbanks

1. Download `ReadSonar.py` and `sonar_img.py` from https://github.com/peterguzw0927/Senior_Design

4.3. Edge Detection

1. Download `estimation.py` and `FcnDetect.py` from https://github.com/peterguzw0927/Senior_Design

4.4. Sonar SLAM 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 `lawnmower_hovering.py`, which tests the HoloOcean side-scan sonar simulation on a vehicle with a lawnmower navigation path

5.2. Basurelle Sandbanks

1. Run `sonar_img.py` in the terminal, which reads multiple .xtf files from the sandbanks datasets.

5.3. Edge Detection

1. In the terminal:
   - ➢ cd to the path containing the Python files with the corresponding image.
   - ➢ Run `python estimation.py`
2. The code tests on a sample sonar image and produces a final filtered image, highlighting the contours, and position coordinates of detected landmarks.

3. It will update the vehicle location based on a Gaussian model.

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 the image and save the data after completion
4. Calculate Map Error and Travel Distance to Achieve Designated %'s of Coverage in the Marina Environment
5. Compare the Error with other types of slam algorithms in the simulation data provided.

6. Measurable Criteria

6.1. Holoocean

1. Produces a reasonable and usable image output

6.2. Basurelle Sandbanks

1. The code should be able to read .xtf files and generate sonar images.
2. It should return the latitude, longitude, depth, and heading of the vehicle.

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 >= 80%.
3. Output the number of landmarks on sonar images.
4. Output the geographical coordinates of each landmark.
5. Use the Gaussian model to estimate the correct vehicle location.

6.4. SLAM Algorithm

As there is no ground truth information available in our real-world experiments, we have instead created the simulated map of Fig. 2 using manually collected data from the harbor environment used in those experiments. Mapping and localization errors are reported with respect to our experimentally derived ground truth point cloud map. The mapping error is computed as follows: For every estimated feature point, we compute its distance to the nearest ground truth point.

7. Test Results

7.1. HoloOcean

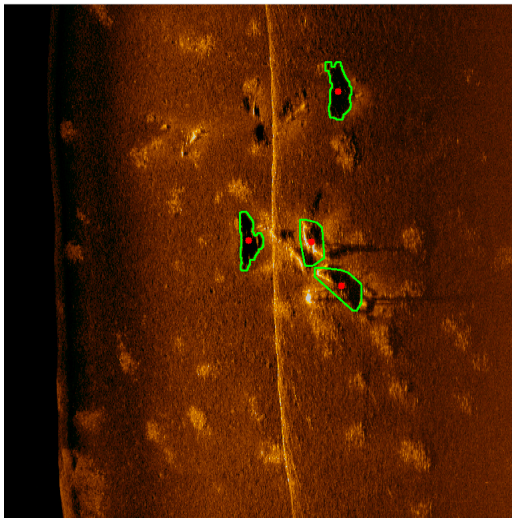
We are now able to navigate the robot correctly in HoloOcean, however, we experience issues due to code that is incompatible with the SCC (it likely uses something along the lines of `os.environ["CUDA_VISIBLE_DEVICES"]="0"`, which is not functional with SCC's many GPUs). Furthermore, we are currently unable to have the robot maintain a constant altitude from the seafloor and are stuck using a constant depth instead. This is not ideal, but still usable for our purposes. Ideally, we'd like to fix this in the next few weeks.

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.



7.4. SLAM Algorithm

Table1: Pose Uncertainty In the Simulated Marina Environment (M^2)									
Distance(m)	0	50	100	150	200	250	300	350	400
Blue-rov Slam(sim)	0.10	0.34	0.30	0.32	0.33	0.31	0.31	0.29	0.28
NF	0.10	0.33	0.44	0.43	0.46	0.45	0.40	0.39	0.40

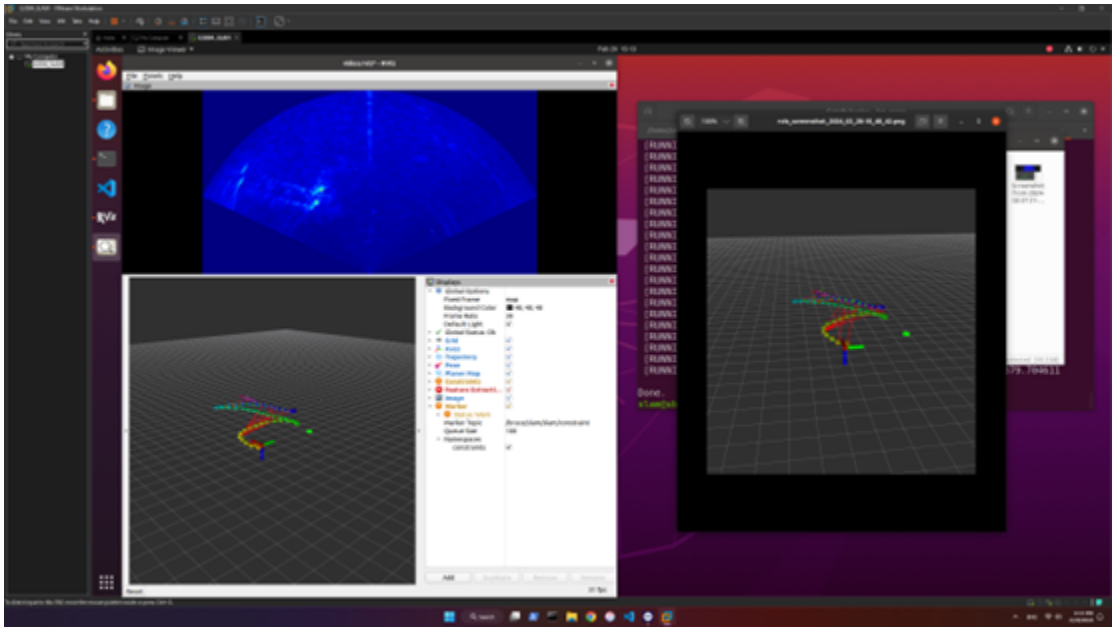
NBV	0.10	0.35	0.53	0.57	0.53	0.45	0.36	0.36	0.35
Heuristic	0.10	0.34	0.32	0.30	0.31	0.30	0.31	0.28	0.31

Table2:Pose Error in Simulated Marina Environment (M)									
Distance(m)	0	50	100	150	200	250	300	350	400
Blue-rov Slam(sim)	0.75	1.72	1.83	1.89	1.90	1.93	1.93	1.96	1.97
NF (sim)	0.75	1,79	1.94	1.97	2.06	2.07	2.08	2.07	2.08
NBV (sim)	0.75	1.90	1.90	1.99	2.07	2.05	2.05	2.05	2.05
Heuristic (sim)	0.75	1.84	1.84	1.88	1.91	1.94	1.95	1.96	1.98

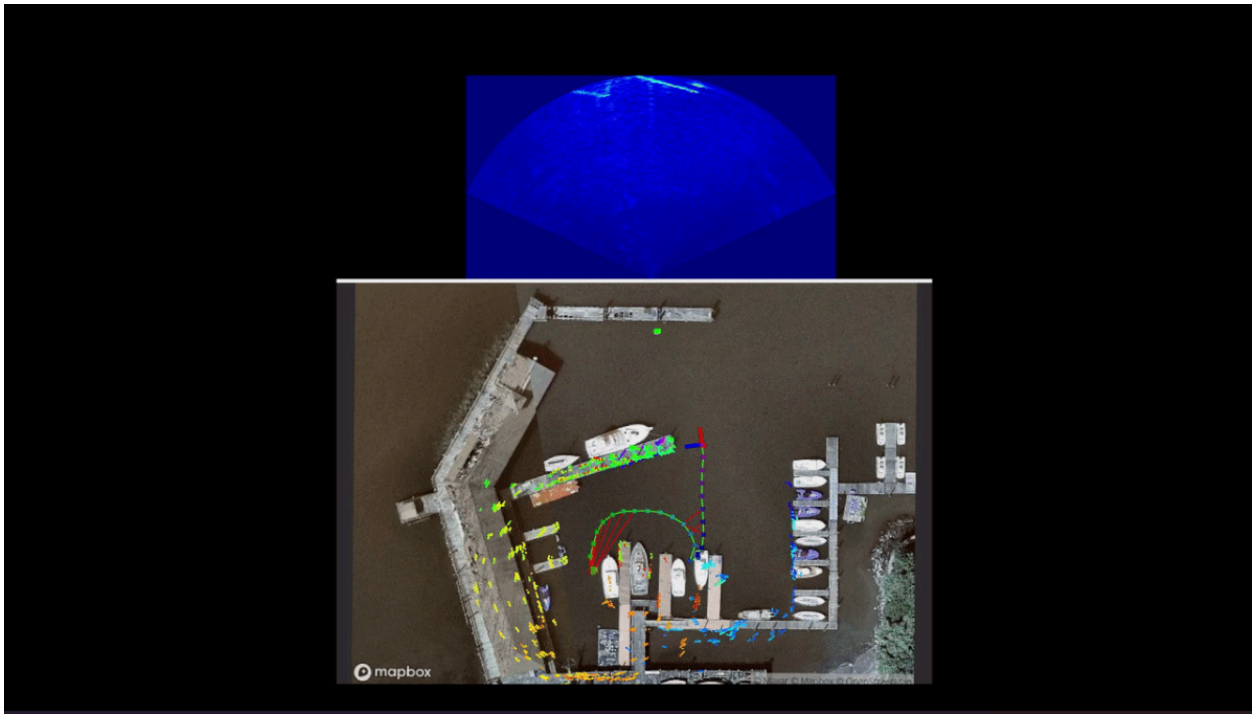
Table3:Map Error in Marina Environment (M)									
Distance(m)	0	50	100	150	200	250	300	350	400
Blue-rov Slam(from data)	0.57	0.95	0.96	0.96	1.04	1.05	1.07	1.06	1.10
Blue-rov Slam(sim)	0.56	0.90	0.95	0.97	1.00	1.03	1.05	1.06	1.05
NF (sim)	0.60	0.97	1.03	1.08	1.13	1.15	1.12	1.13	1.13
NBV (sim)	0.58	0.87	1.01	1.08	1.12	1.12	1.12	1.12	1.12
Heuristic (sim)	0.58	0.90	0.94	0.98	0.99	1.01	1.02	1.03	1.03

Table4: Travel Distance to Achieve Designated %'s of Coverage in the Marina Environment				
Coverage	50%	60%	70%	80%

Blue-rov Slam(from data)	13.79	39.53	80.92	157.15
Blue-rov Slam(sim)	14.88	41.98	77.49	176.31
NF(sim)	14.90	43.63	80.92	268.67
NBV(sim)	13.90	40.55	88.38	234.01
Heuristic(sim)	14.23	41.41	80.12	396.52



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

- Since the last round of testing, we have reached out to the Frost Lab at Brigham-Young University, which is responsible for the development and maintenance of the HoloOcean simulator. With their assistance, we have been able to write a simulation script that navigates the robot to our liking. However, there are currently compatibility issues with the SCC that we must address to achieve the desired results.

8.2. Basurelle Sandbanks

- This testing is to demonstrate what a typical sonar dataset looks like and what measurements can be taken from such a dataset. Our code successfully rectifies the sonar datasets into images and obtains the geographical location of the vehicle at each ping.

8.3. Edge Detection

- The current edge detection algorithm is reliable because it can successfully detect the correct landmarks on multiple sonar images. It's also able to determine the landmark coordinate with respect to the vehicle. By incorporating this algorithm with the sonar images and position coordinates of the vehicle extracted from the sonar dataset, we could generate a global map storing both the locations of the landmarks and the vehicle. In addition, it's able to update the vehicle location when it revisits the same area based on the mean and covariance of the Gaussian model, which is very promising.

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.)
- The pose uncertainty, trajectory error, and map error of the heuristic and Blurov algorithms are similar in value but significantly lower than that of the NF and NBV algorithms. While the heuristic and Bluerov algorithms have similar pose uncertainty and map/trajectory error, the heuristic approach falls behind Bluerov in terms of map coverage -> Favors exploration but still did a good job of keeping the error small.

In a nutshell, we achieved the total functional requirements of the project based on the aspect of completeness, and integration of different modules. We successfully and qualitatively compare our algorithm with other existing algorithms with multiple metrics (Pose Uncertainty, Map Error, Pose Error, and Travel Distance to Achieve a Designated percentage of Coverage).

9. Appendix

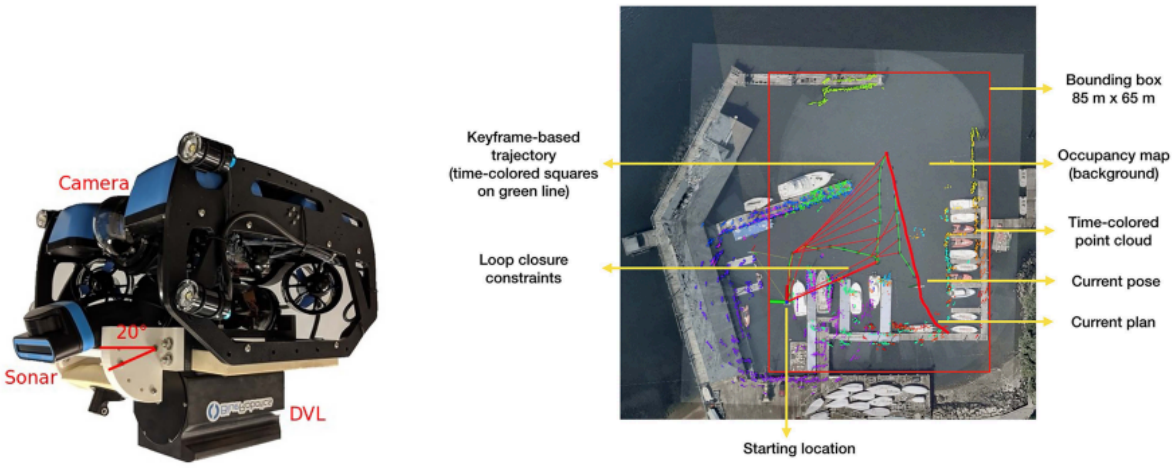


Figure 1. Custom-instrumented BlueROV2 robot used in experiments, and an illustration of the setup of experiments at the U.S. Merchant Marine Academy in King's Point, NY, USA, right is the satellite image.

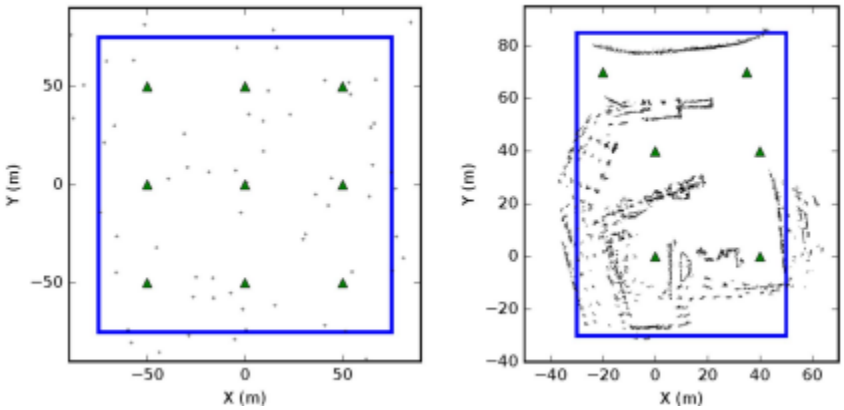


Figure 2. Simulated exploration environments for landmarks and pose SLAM. The exploration bounding box is denoted by a blue rectangle, and exploration is initialized from the green triangles. The robot may only travel within the bounding box, and only observations of the area within the bounding box are used to evaluate map coverage.

Table data reference: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9806387