Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Prototype Testing Report

by

Team 16
The Sharks

Team Members

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: Nov 19

1. Abstract

The absence of available underwater sonar datasets before Week 10 prompted the creation of our own landmark simulation for evaluation. This report details three distinct approaches utilized for underwater SLAM testing. The first involves a rudimentary Python-based underwater world simulation, enabling error estimation for SLAM algorithms based on provided landmark data. The second employs the HoloOcean Simulator that can create an underwater environment for collecting practical sonar data, while the third directly deploys ORB-SLAM3 using sample datasets.

2. Required Materials

2.1 Hardware:

- Working PC (preferably Windows)
- GPU - Currently using Nvidia Tesla V100 on SCC

2.2 Software:

- Python 3.0 libraries
    - NumPy
    - matplotlib
    - seaborn
    - Pandas
    - HoloOcean
- C++ libraries
    - OpenCV: >= 3.0
    - boost
- Cmake: >= 2.8
- Visual Studio: >= 2015

3. Set up:

Since this is a software-only project, most of these tests can be entirely completed on a working computer. The exception is the HoloOcean simulator, which we have only been able to run on machines with a strong GPU. Today, we will use the V100 GPU available on the BU SCC.

Both HoloOcean and the basic SLAM algorithm were written with Python scripts, so Python and all required libraries must be installed and on your path environment variable. ORB-SLAM3 will require CMake and Visual Studio to build the packages and the program as all codes were written in C++. During the testing of ORB-SLAM3, the program will localize the position of the camera and perform loop closure to update the mapping when creating a visualized map.

Basic SLAM:

Create a world with desired parameters. The parameters are confined to a simple and comprehensive range.

```
# world parameters
num_landmarks      = 5       # number of landmarks
N                  = 20      # time steps
```

```
world_size         = 100.0    # size of world (square)
```
Then we need to initialize desired robot parameters
```
measurement_range  = 50.0     # range at which we can sense landmarks
motion_noise       = 2.0      # noise in robot motion
measurement_noise  = 2.0      # noise in the measurements
distance           = 20.0     # distance by which robot (intends to) move each iteration
```

4. Pre-Testing Setup Procedures:

4.1 Holoocean:

1) Connect to Desktop on SCC with V100 GPU
2) Run the following set of commands to prepare HoloOcean:
    a) `cd /projectnb/ece601/SlamSeniorProj2023`
    b) `module load python3/3.10.12`
    c) `source hocean/bin/activate`
    d) `export HOLODECKPATH=/projectnb/ece601/SlamSeniorProj2023`
3) Run HoloOcean files using `python <<filename>>`
    a) Will test both side scan and forward look sonar

4.2 Basic SLAM:

1) Run the Python script demo.py

4.3 ORB-SLAM3:

1) Git clones the repository https://github.com/chanho-code/ORB-SLAM3forWindows.git
2) Ensure all the prerequisite packages (OpenCV, boost) and tools (Cmake, Visual Studio) are installed.
3) Ensure all installed packages are added to the system path.
4) Make a 'build' directory for each package in the "Thirdparty" directory of ORB-SLAM3 for Windows and configure and select Visual Studio 14 2015 x64 (can be another version of Visual Studio).
5) Use Cmake to build the required packages (DBoW2, g2o, and Pangolin) for ORB-SLAM3.
6) For the pangolin package, make sure not to disable BUILD_EXTERN_GLEW, LIBJPEG, LIBPNG. Continue to open the project in Visual Studio even if warning messages existed during the generation process.
7) When building each package in Visual Studio, change the build type to release.
8) Navigate to the project of each package -> Properties -> General -> Target Extension: .lib, and Configure Type: Static Library (.lib)
9) Change the runtime library of the project to "Multi-threaded(/MT)"
10) Build All_BUILD. (ignore "cannot open input file 'pthread.lib' for pangolin after the build is done)
11) After you finish building all 3 required packages, repeat the same steps to build the "ORB-SLAM3 for Windows" package as you did to the other packages using Cmake and

Visual Studio. Also, include *'COMPILEDWITHC11'* in preprocessor definitions under project -> Properties -> C/C++ Tab -> Preprocessor.

12) For the “ORB-SLAM3 for Windows” package, build “ORB-SLAM3” project only (not ALL_BUILD).
13) To build the test project “mono_inertial_euroc”, repeat step 11 to have the same settings as “ORB-SLAM3” project. In addition, add the path of the boost library to this project and define an ‘usleep' function in mono_inertial_euroc.cc.
14) Build “mono_inertial_euroc”.

<u>**5. Testing procedure:**</u>

<u>5.1 Holoocean:</u>

1) Run `example.py`, which tests the HoloOcean side-scan sonar simulation
2) Run `profile_sonar.py`, which tests the HoloOcean forward-look sonar simulation

<u>5.2 Basic SLAM:</u>

1) Uncomment test case 1
2) Run test case 1
3) Compare estimated landmarks and poses with actual landmarks and actions made
4) Calculate the error (done by program)
5) Uncomment test case 2
6) Run test case 2
7) Compare estimated landmarks and poses with actual landmarks and actions made
8) Calculate the error (done by program)

<u>5.3 ORB-SLAM3: (EuRoC MAV Dataset)</u>

1) The test program can be found in *path_to\ORB-SLAMforWindows\Examples\Monocular-Inertial\Release*
2) Download the machine hall 01 dataset (ASL format) using the following link https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets
3) Make sure to unzip the dataset and put it into a new directory named MH01.
4) Extract ORBvoc.txt from ORBvoc.txt.tar.gz in *path_to\ORB-SLAMforWindows\Vocabulary*
5) On the command prompt, cd to the directory containing the test program.
6) Execute: *mono_inertial_euroc.exe path_to\Vocabulary\ORBvoc.txt path_to\Monocular_Inertial\EuRoC.yaml path_to\MH01\ path_to\Monocular_Inertial\EuRoC_TimeStamps\MH01.txt dataset-MH01_monoi*
7) Images and IMU data will be loaded into the program.
8) A map viewer window will pop up along with the video recording of the surrounding environment using a camera.
9) Compare the motion of the keyframes in the map viewer and the motion of the camera.

<u>**6. Measurable criteria:**</u>

<u>6.1 Holoocean:</u>

1) Produces a reasonable and usable output

6.2 Basic SLAM:

I. The SLAM algorithm can read a given input describing an environment
II. After simulating movement with noise, the algorithm can produce an estimation of a simulated robot location relative to a set of landmarks

Desired error thresholds for Landmark and Pose Errors:

- Landmark errors should be minimal. Since the model is basic, the error should preferably be less than 0.5, below 0.1 is perfect, and below 0.3 is good.
- Pose Error is more tolerable since uncertainties accumulate after multiple iterations. The preferable mean error is less than 1, below 0.3 is perfect, and below 0.5 is good.

Test1	Percentage Error%
Mean Pose Error	0.238
Mean Landmark Error	0.152
Test2	**Percentage Error%**
Ending Pose Error	0.241
Mean Landmark Error	0.147

6.3 ORB-SLAM3: (EuRoC MAV Dataset):

For this algorithm, the discernible criteria involve the successful execution of the test, specifically ensuring that the simulated robot accurately identifies landmarks using the camera and appropriately visualizes the map points on a distinct display window. The motion of the keyframes in the map viewer should align with the motion of the camera while operating in real time.

Test	Percentage Error%
Simulation	0%, the simulation was correctly made because the motion of the keyframes matched the motion of the camera

7. Conclusions:

7.1 Holoocean:

The test results strongly support integrating HoloOcean into our project. A close comparison between client data and HoloOcean-generated images reveals significant similarity. This alignment underscores HoloOcean's reliability and suggests its potential to enhance our project's visual aspects effectively. Moving forward with HoloOcean integration seems promising for improving data representation and analysis.

7.2 Basic SLAM:

I. Test 1 exhibited excellent performance, with a mean pose error of 0.23, categorizing it as perfect since it is below the threshold of 0.3. Additionally, the mean landmark error recorded at 0.15 falls within the range of 0.1 to 0.3, classifying it as consistently good.
II. Test 2: The test results had a 0.24% mean pose error and a 0.14% mean landmark error. This means that the error falls within the range of consistently good as well.

7.3 ORB-SLAM3: (EuRoC MAV Dataset):

Given its proficiency in creating an accurate map view by processing the images and IMU (inertial measurement unit) data from a micro aerial vehicle (MAV), ORB-SLAM3 demonstrates its values as a guiding benchmark in the development of our own SLAM algorithm. The next crucial steps involve testing its adaptability and performance using our client's dataset with ORB-SLAM3. However, since our data are unique sonar data collected from underwater vehicles as there's only INS (Inertial Navigation System) data instead of IMU data, it might be a challenge for us to directly feed the data into ORB-SLAM3. Fortunately, it is possible to test only the images created by the sonar data with ORB-SLAM3, but we predict that the accuracy rate could be lower due to missing IMU data. Despite this challenge, we remain committed to exploring the content and logic within ORB-SLAM3 because the tracking and relocalization methods employed by ORB-SLAM3 to map the vehicle in 3D space have proven to be efficient and helpful. We will also continue to explore potential methods to address the replacement of IMU data with INS data due to the unique characteristics of our sonar dataset.

Overall, this finding marks a strategic move towards tailoring the algorithm to achieve our specific requirements through a comprehensive evaluation of its efficacy in a real-world context. We aim to develop a successful SLAM algorithm for our sonar dataset that can mirror or surpass the efficiency and accuracy rate demonstrated by ORB-SLAM3.