EECE5554-Robotic Sensing and Navigation

Final Project Report

Mapping with ORB-SLAM3

Group 6

Sadik Bera Yuksel

Haoxin Liu

Hrigved Suryawanshi

Mihir Chaulkar

Challenges

- **Installation Issues and Resolutions for Eigen3 and ORB-SLAM3**

  Issue 1: The standard installation command ("sudo apt-get install libeigen3-dev") for Eigen3 does not place the library in the correct directory for ORB-SLAM3 detection. Using this command installs Eigen3 to /usr/include/eigen3, but ORB-SLAM3 fails to detect it there, leading to compilation errors. Instead, installing Eigen3 from source code ensures it is placed in /usr/local/include/eigen3/, which is the directory ORB-SLAM3 checks.

  Issue 2: The command ./build.sh from the original repository is designed to build the entire environment and its packages concurrently. However, this process is significantly slow and prone to numerous errors. One error we faced was related to the OpenCV version; ORB-SLAM3 requires a minimum of OpenCV 4.0. To resolve this, we edited CMakeLists.txt file to specify the OpenCV version installed on our system. Another issue we faced was related to the "realsense2config.cmake" file. To fix this issue we had to install the necessary ROS package with "sudo apt-get install ros-melodic-realsense2-camera" command.

  Issue 3: build.sh process is terminated due to Loop.cc file misconfiguration. A compilation termination error can occur due to incorrect C++ configuration settings in ORB-SLAM3. We solved this issue by adding the following line at the start of the CMakeLists.txt file: "add_compile_options(-std=c++14)". This ensures that the C++14 standard is used, allowing all files to meet the installation requirements and complete the compilation process.

- **IMU initialization:** As it is mentioned in the ORB-SLAM3 paper, the IMU initialization should include IMU roll/pitch rotation. At the beginning of the data, the camera should move with enough translation and the same time, enough roll/pitch rotation for IMU initialization. However, we did not have any roll/pitch rotation initially in NUance dataset as the car moves on a flat surface.

<u>Results</u>

We used "KRI_vision_IMU_GPS" data set and a walking video we recorded with a phone camera during our project. For ORB-SLAM3 to work properly, a YAML file that contains the camera and IMU sensor parameters should be provided in the correct format. However, the YAML file provided with the dataset was not in the correct format so we created another one manually by using the same parameters in the given file. For the phone camera, we used EuRoC dataset YAML file for monocular cameras.

- **NUance Dataset**

We first did some trials with ORB-SLAM3 Visual-Inertial mode on "KRI_vision_IMU_GPS" dataset but the algorithm failed to initialize the IMU and kept losing track due to the reasons we mentioned in the Challenges section. This resulted in frequent map resets and the algorithm could not find a complete trajectory. Link to the Stereo-Inertial mode demo: https://www.youtube.com/watch?v=U86LykdYIaw

After that, we decided to use Stereo mode on "KRI_vision_IMU_GPS" dataset. When we played the Rosbag at its original speed, the algorithm again lost track and failed to localize the camera when the car made turns. Therefore, whenever it lost track, it started from the beginning and created a new path. The resulting trajectory is as follows:

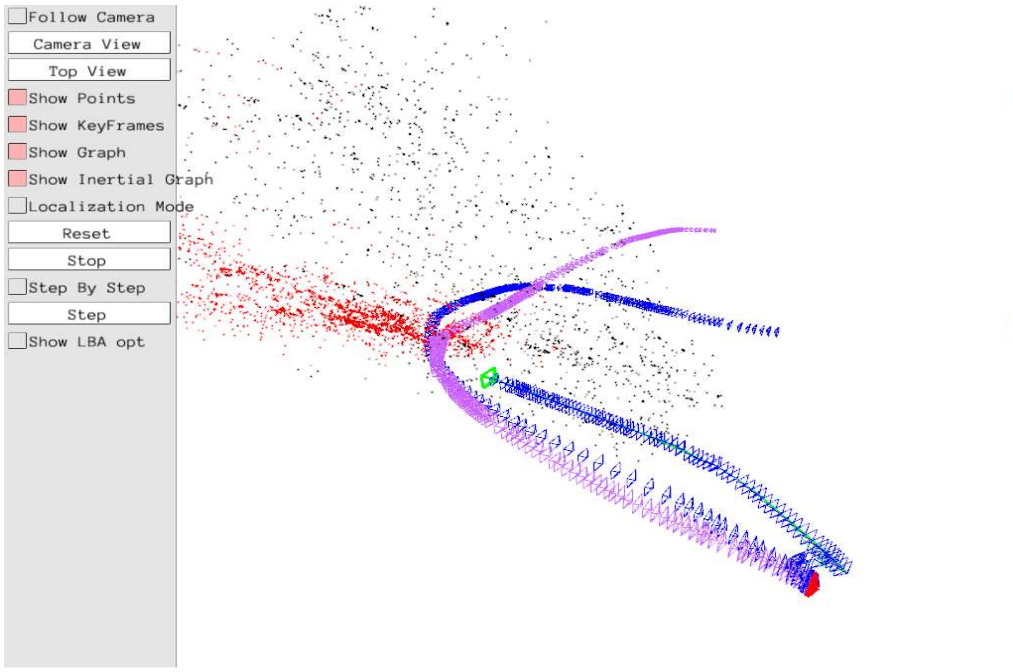
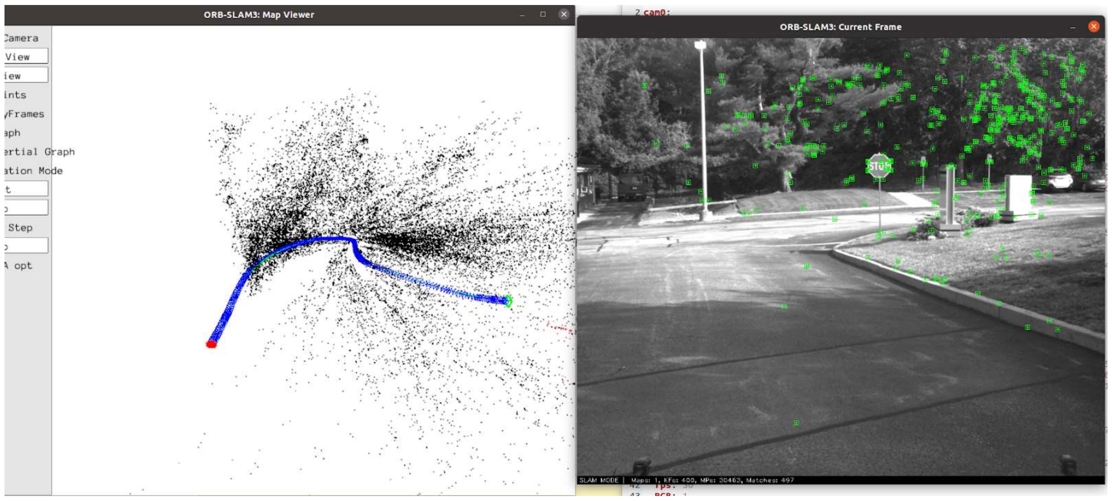


Figure1. Trajectory generation in Stereo Inertial mode when we play Rosbag at the original speed

To resolve this issue, we played the Rosbag at a lower speed (x0.3) to give the algorithm enough time to find the matching features between consecutive frames when the car makes fast turns. This enabled the algorithm to keep track during the turns and we successfully created the trajectory. The link to the Stereo mode demo:
https://youtu.be/XyGDZieq8Mw?si=dBmc7dkmqq5wVSEF



```
SLAM MODE |  Maps: 1, KFs: 400, MPs: 30463, Matches: 497
```

Figure2. Trajectory generation in Stereo mode when we play rosbag at x0.3 speed.

These two trajectories (Fig.1 and Fig.2) demonstrate how the timing of sensor data can impact the SLAM process. The system's ability to track and map depends on the expected rate of incoming data. We found out that if data arrives too quickly or slowly, it might miss features or misinterpret motion, leading to divergent paths like those shown in Fig.1. These visual comparisons are essential for fine-tuning the parameters of SLAM systems and for understanding the limitations of the current setup under varying operational conditions. We also found out there were some timeframes that the features were too far to detect which was also mentioned by the original author. So when we reduce the speed of play for Rosbag, the system will have a better reaction to detect the features more accurately and smoothly. The final trajectory generated by ORB-SLAM3 and the ground truth trajectory obtained by GPS data are given below:

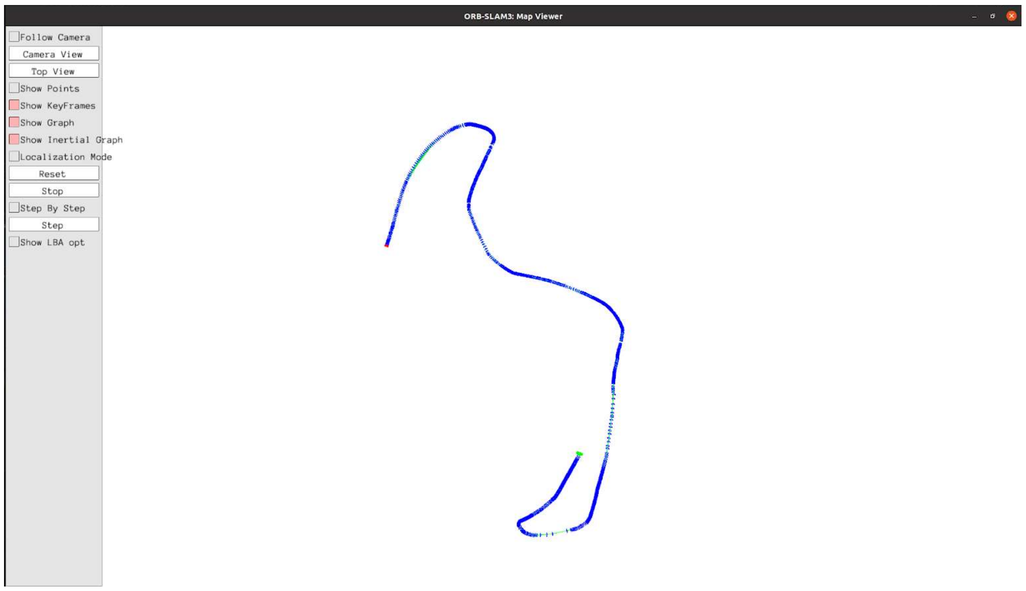


Figure3. Final trajectory generated by ORB-SLAM3 Stereo Mode using NUance data

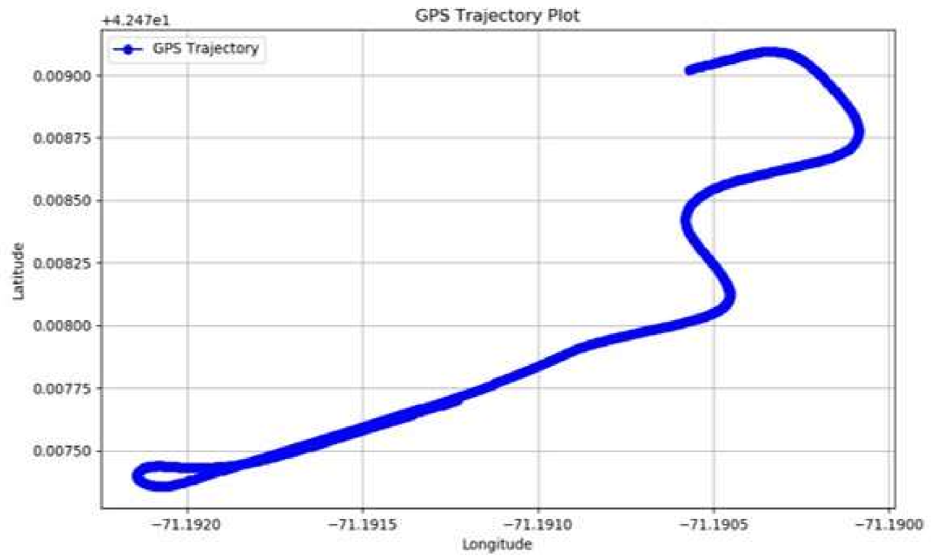


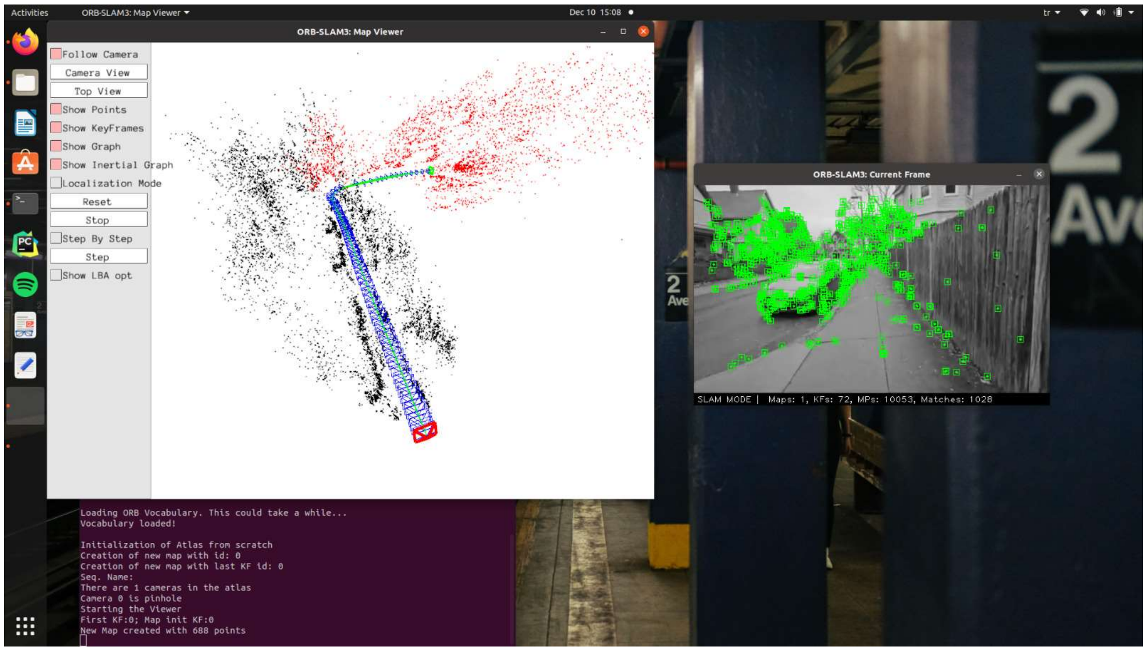
Figure4. Trajectory generated by GPS using NUance data

In our comparative analysis, we noticed minor discrepancies toward the terminus of the plotted paths, with the GPS data illustrating a closed loop that is not as discernible in the ORB-SLAM3 trajectory representation. One plausible explanation for this variance could be attributed to the intrinsic differences between two-dimensional and three-dimensional visualization techniques. ORB-SLAM3's representation may not translate certain spatial nuances effectively into 2D space, which can obscure details such as closed loops. Moreover, ORB-SLAM3's performance can be contingent on the quality of keyframes selected during the SLAM process. These key frames are crucial as they anchor the system's understanding of the environment; however, if these frames capture dynamic objects, such as moving vehicles, the accuracy can be compromised. The system's performance in two dimensions is generally robust, but it can falter when translating complex three-dimensional movements and structures into a 2D plane, especially when those movements involve intricate paths like loops or when the keyframes are affected by transient, dynamic objects.

- **Self-collection data by phone:**

In this data collection, we performed an outdoor walk in a residential neighborhood, meticulously capturing a video to document the route taken, as shown in Figure 7. We then used a custom Python script to convert the video into a format compatible with ROS, specifically into a rosbag file. This conversion process involved the utilization of the cv2.VideoCapture function to sequentially capture each frame from the video. Each frame was then converted into an image message using the cv2_to_imgmsg function, facilitating their publication within the ROS ecosystem. We configured ORB-SLAM3 to operate in monocular mode. This decision acknowledges the smartphone's limitation to a single camera setup. As mentioned before, we used EuRoC dataset YAML file for monocular cameras for the camera calibration parameters required by ORB-SLAM3.

Moreover, we paid particular attention to the system's loop closure capability, which is pivotal for correcting any cumulative drift that may occur as the camera navigates through the environment. Screenshots demonstrating the generation of the rosbag and the subsequent trajectory—including points where loop closure was successfully identified and applied—are included to provide a visual account of the system's performance during the data collection.



```
SLAM MODE |  Maps: 1, KFs: 72, MPs: 10053, Matches: 1028
```

```
Loading ORB Vocabulary. This could take a while...
Vocabulary loaded!

Initialization of Atlas from scratch
Creation of new map with id: 0
Creation of new map with last KF id: 0
Seq. Name:
There are 1 cameras in the atlas
Camera 0 is pinhole
Starting the Viewer
First KF:0; Map init KF:0
New Map created with 688 points
```

Figure5. Trajectory generation by ORB-SLAM3 using our own collection data set

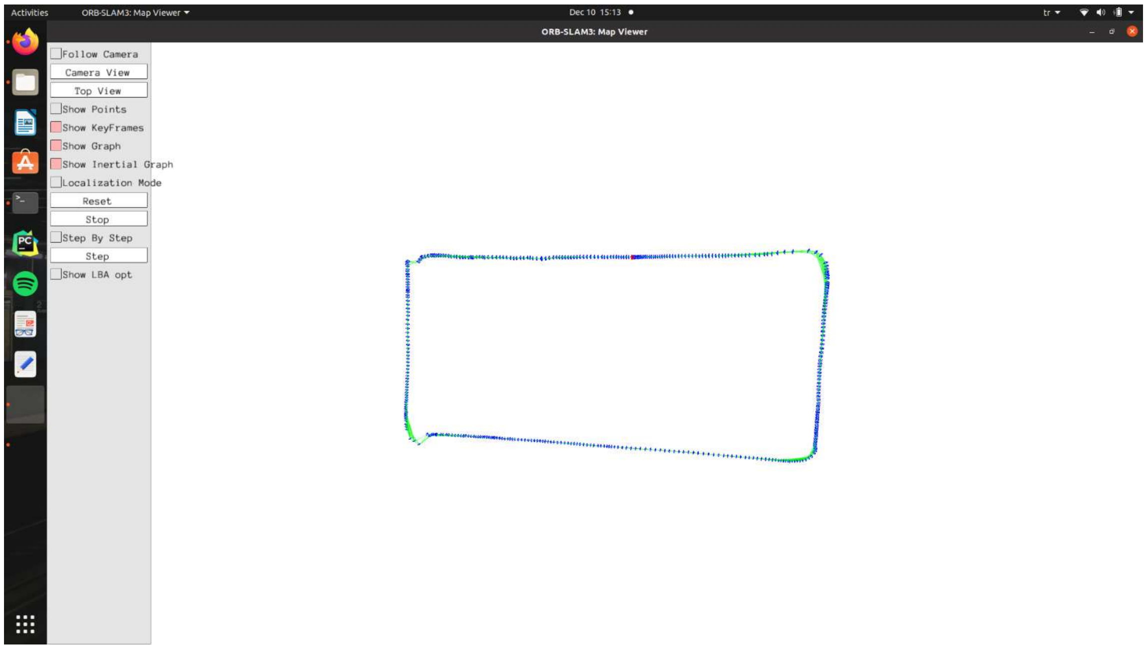


Figure6. The full trajectory generated by ORB-SLAM3 using our collection data set

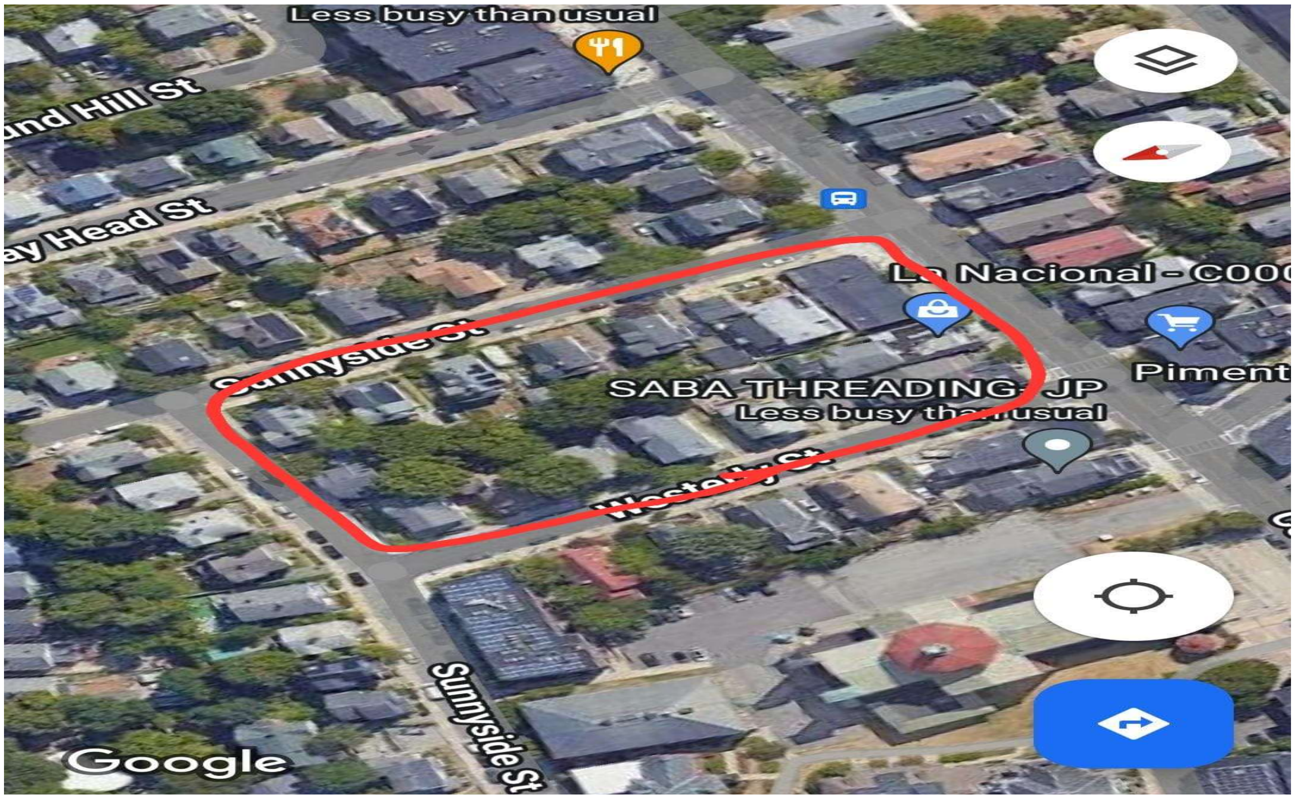


Figure7. The actual trajectory of the walking data from Google Maps

By comparing the trajectory made and the actual path on Google Maps. We observe a high degree of similarity, with both paths approximating a rectangular shape. The trajectory generated by ORB-SLAM3, as seen in Figure 6, appears to closely mirror the true path that we traversed, which is represented in Figure 7.

This successful mapping can be attributed to several factors during the data collection phase. The place where we walked allowed for the ORB-SLAM3 system to effectively capture and process each frame with a high degree of positional accuracy. Slow movement reduces the motion blur in images and ensures that each frame provides a clear view of environmental features necessary for accurate tracking and mapping. Secondly, the smoothness of our turns played a pivotal role. Sudden or sharp turns can introduce estimation errors in SLAM systems; however, by navigating the turns gently, we minimized these potential inaccuracies.The loop closure functionality works by recognizing previously visited locations in the video, allowing the system to realign the current trajectory with past segments, thus enhancing the accuracy and reliability of the mapped path.