EECE 5554 Robot Sensing and Navigation - Final Project Report

Exploring Various SLAM Algorithms

Bibek Gupta, Nathan McGuire, Azhar Syed, Tae Yang

Introduction

SLAM, which stands for Simultaneous Localization and Mapping, is a critical technology used in robotics, augmented reality, and other fields that require precise positioning and mapping in real-time. The goal of SLAM is to enable a robot or other device to navigate an environment and construct a map of that environment simultaneously, using only sensor data. With SLAM, machines can navigate and interact with the environment more accurately and effectively, making it a crucial component of many modern technologies. In this project, different lidar and visual slams were explored such as Hector SLAM, Gmapping, Cartographer, RTAB-Map and ORB-SLAM 3 with a goal to create a map or trajectory of a rount-trip route. Sensor data set was collected using an IMU, Realsense Cameras, and Lidar and run on the SLAM algorithms mentioned earlier. Finally, the trajectories or maps created by each SLAM algorithm were compared and challenges with these SLAM algorithms on our data set were discussed.

Data Collection Setup

Map data were collected from four sensors, which were rigidly attached to a wheeled cart, and pushed through the tunnels between Forsyth and Churchill Halls. The round-trip route taken in the tunnels is indicated as a red line in Fig. [1].

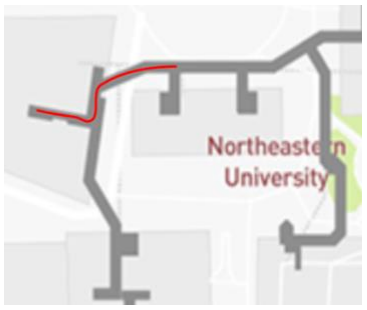

Figure [1]: Route and map of the tunnels

The sensors used were two camera systems, a Realsense T265 and a Realsense D435i, a VectorNav V100 IMU, and a RPLIDAR A3 2D planar LiDAR sensor as shown in Fig. [2].

Figure [2]: Sensor images of RPLIDAR A3, Realsense T265, D435i, and VN100 from the left.

The RPLIDAR A3 is a two-dimensional planar lidar. It has a single laser rangefinder mounted on a platform that spins 11 times per second. 1947 per revolution it reads the range from the laser rangefinder. This results in a new range measurement every 0.185 degrees of rotation. Since the platform spins around the vertical axis of the sensor, the ranges are all along a plane parallel to the ground at the elevation of the sensor. The Intel Realsense T265 is a black and white, stereo, fisheye camera system that does onboard visual odometry. An onboard processor reads in images and runs the feature extraction, matching, and pose estimation, allowing the user to offload that computation to an external device. The camera provides the user with the image streams and odometry measurements. The Intel Realsense D435i is an infra-red projection based RGBD camera system. It contains an IR stereo pair, an IR projector, and an RGB camera. The IR projector broadcasts a known pattern onto the world, which the stereo pairs read back. The stereo processing algorithm uses the reflected IR pattern from each camera to determine the parallax of each portion of the image and compute the distance from that. The RGB image is then overlayed on top of the depth image to create the full RGBD image data. The VectorNav VN100 is a high quality, 9 axis, MEMS AHRS. It contains a three-axis accelerometer, gyroscope, and magnetometer, as well as an onboard processing unit to perform sensor fusion.

The sensors were rigidly attached to a rolling cart, to ensure that none of the sensors moved relative to each other during data collection. All four sensors were connected to a single laptop, which ran the ROS drivers for each device, as well as a rosbag node. Once all the sensors were providing good data, and the rosbag was recording them, the cart was pushed through the tunnels. The cart started outside the Mechanical Engineering Capstone Lab, and moved towards Churchill Hall, where it turned around and drove the same route back to the starting point.

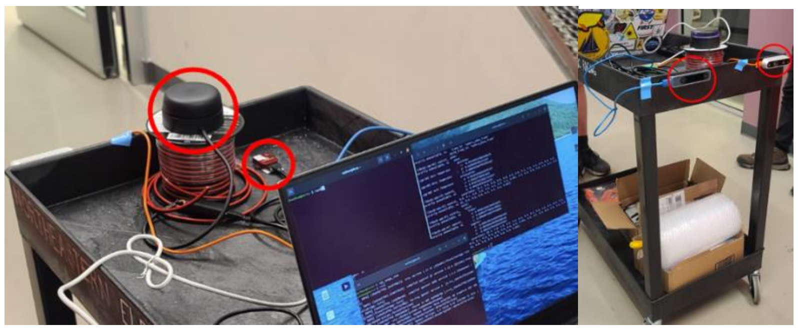

Figure [3]: Top side of the cart with the lidar and IMU sensor and front side of the cart with the cameras

Hector SLAM

Hector SLAM is a ROS package implementing a LiDAR based SLAM algorithm. It gets its name from the Hector Unmanned Ground Vehicle that the algorithm was initially demonstrated on, since the paper authors neglected to name it in their paper. This approach implements a LiDAR only SLAM approach that matches each new incoming scan to the map. Scan matching is performed by finding the sensor pose that minimizes the error between the map and the new data using the Gauss-Newton least squares approach. This means that no feature detection or matching is required, which reduces the computational requirements of this approach.

Hector slam worked well most of the time. It accurately generated and expanded the map as the cart moved along and provided good position information. However, when the cart was brought into the long, straight sections of the hallway, Hector slam was unable to function properly. The LiDAR could only see two straight walls, scan after scan looked identical, so the scan matcher decided that the cart was not moving. Once it got to a bend in the hallway, or a section with any geometry at all, Hector slam resumed tracking position. However, large sections of hallway got “missed” while the algorithm thought the cart was not moving, so the map was very wrong by the end.

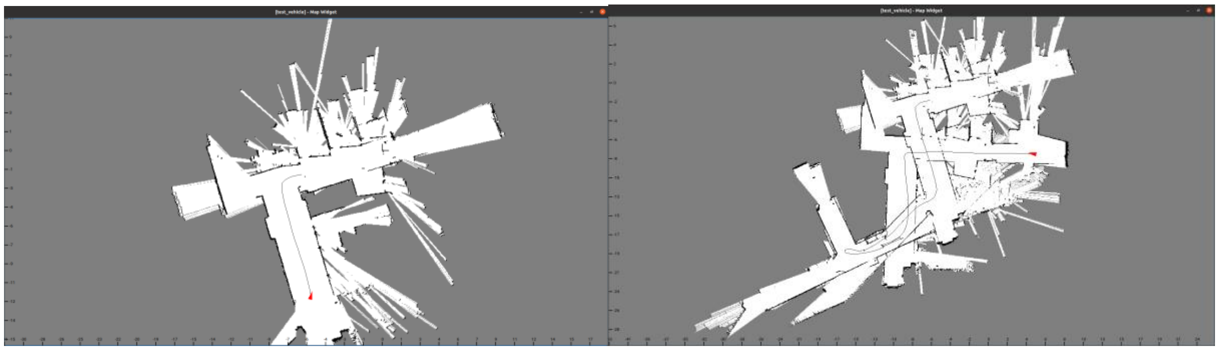

Figure [4]: Maps generated from Hector Slam. Left: First 30 seconds of dataset, showing good performance with sufficient geometry. Right: Full map showing drift.

Gmapping SLAM

Gmapping SLAM also provides laser-based SLAM. It applies a Rao-Blackwellized particle filter for learning grid maps. In ROS, the gmapping package has a node called slam_gmapping which creates a 2-D occupancy grid map from laser and pose data. The node attempts to transform each incoming scan into the odom tf frame.

When run with our rosbag data, the pose data were provided by another node called laser_scan_matcher_node from the laser_scan_matcher package. This package is an incremental laser scan registration tool. It allows to scan match between consecutive sensor_msgs/LaserScan messages and publish the estimated position of the laser as a tf transform.

The resulting map is shown in Figure [5]. The laser scan matcher node worked well when there were distinctive differences between consecutive laser scan messages. When the data collecting cart was driving in straight lines, the pose of the cart was assumed to be stationary because the laser scan matcher node could not find a registration between the two consecutive straight line laser scans. Therefore, some tunnel maps were generated on top of the wrong locations and the resulting map had overlapping areas.

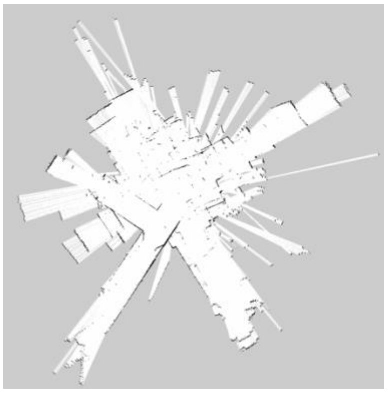

Figure [5]: Map generated from Gmapping SLAM

Cartographer SLAM

Cartographer SLAM uses a combination of LiDAR and IMU data to construct 2D or 3D map of the environment while simultaneously estimating the position and orientation of the robot within the map. It is a very convenient slam package in that the user only needs to provide sensor data and the package has well-structured launch files that create all necessary nodes and transformations between different coordinate frames.

The IMU data from the VectorNav IMU was used so that the cartographer node could locate where the cart was accurately in the map. The resulting map is shown in Fig. [6]. Since the IMU data were used, the straight lines were generated more accurately than the past two SLAM algorithms. However, the cartographer node wasn't calculating the pose of the cart accurately when taking turns. The node has an internal imu tracker and scan matching functions to estimate poses from the imu and lidar data. It was found that the pose estimation system was too sensitive to rotations. For example, when the cart rotated 90 degrees, the estimated rotation was about 180 degrees. Therefore, the algorithm couldn't achieve loop closure and the map ended up having a longer path than the actual route.

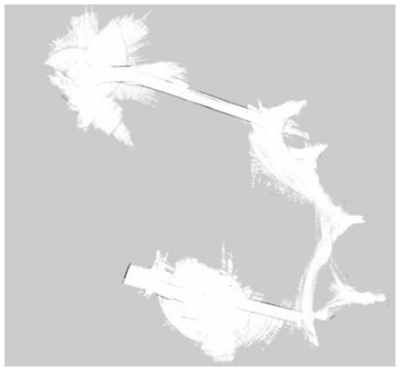

Figure [6]: Map generated from Cartographer SLAM

RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) is an open-source Graph-Based SLAM library for mobile robotics. It is designed to build 3D maps of environments using various sensors such as RGB-D cameras, stereo cameras, and LiDAR sensors. RTAB-Map uses a keyframe-based approach to map building, which means it uses a set of keyframes to represent the environment instead of using a dense point cloud. This approach reduces the memory usage and computational requirements of the SLAM algorithm. It also uses visual and loop closures to improve the accuracy of the map. Visual closures are detected by comparing the appearance of the images captured by the camera, while loop closures are detected by comparing the poses of the keyframes.

The realsense D435i camera data was used to run RTAB-MAP since the slam algorithm had been verified with the camera according to its developers. The resulting occupancy grid map and point cloud map are shown in Fig. [7]. The left image in Fig. [7] shows that both maps were very accurate and identical to the actual route in Fig. [1]. When the cart was coming back at the top right corner in Fig. [7], the path was deviated slightly, but soon RTAB-Map corrected its path after achieving visual closure. The right image shows the inside of the point cloud map. The point cloud accurately represented the tunnel and some objects such as a vending machine and elevator in the tunnel.

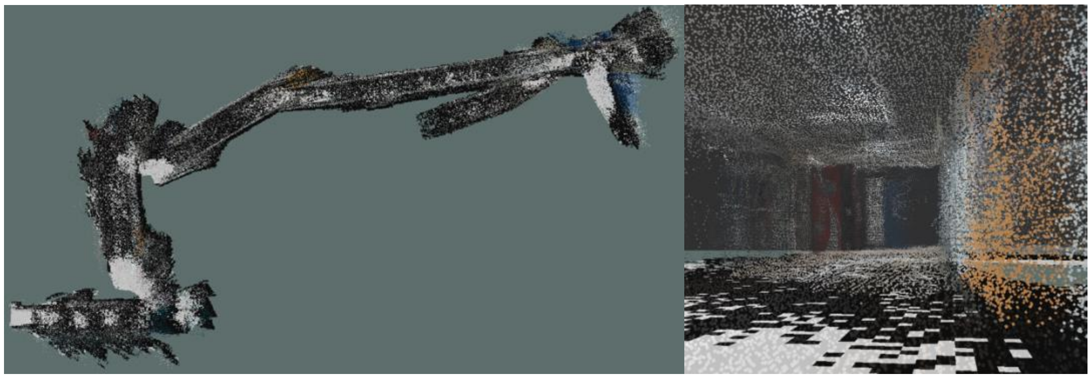

Figure [7]: Maps generated from RTABMAP SLAM

ORB-SLAM 3

The ORB-SLAM 3 is a modern visual SLAM system that is available as open-source software. It can precisely follow the movements and location of a camera in real-time while simultaneously creating a 3D map of the surrounding environment. The ORB-SLAM 3 system is based on the feature-based ORB descriptor and the probabilistic mapping approach of SLAM. In addition, the system is equipped with advanced functionalities, such as loop closure detection, which contributes to improving the map's accuracy over time, and a semantic segmentation component that identifies and categorizes objects within the environment, providing extra information to the system.

Installation and making the ORB-SLAM 3 package work was quite challenging. But after installing the correct dependencies packages' version such as Open CV 4.2, and the latest versions of Pangolin and Eigen, the issue resolves and works very well on KITTI Dataset. The two sensor data from the D435i and T265 camera were used separately. After running the images from RGB-D camera through monocular settings, the ORB-SLAM 3 was not performing well while taking turns and it could happen because while taking turns the images were a little blurry and have some vibrations. To mitigate these issues, various techniques and approaches can be implemented, such as using multiple sensors, optimizing the algorithm parameters, or employing machine learning techniques to improve the system's robustness and accuracy.

Instead, the image data from the fisheye camera were processed under the monocular settings and it was able to localize and map the environment. To get better trajectory, the number of orb features parameters was set to 1000 in .yaml file. Finally, a good visual trajectory was plotted and is shown below in Fig. [8]. The start and end point of trajectory still differs by 3 m which gives percent error of 3% which is quite impressive. The source of error might be because while going to the end point, the cart was on one side of the corridor and while returning it was on other side of the same corridor and the corridor was wide. Another source of error could be the features such as the tiles and color of wall were very similar inside the corridor.

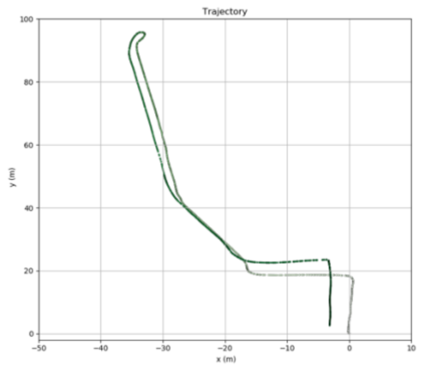

Figure [8]: 2-D Trajectory plot from ORB-SLAM 3

Conclusion

SLAM	Sensors	Map type	Performance	Classification	Based on	Odometry needed
Hector SLAM	2D Lidar	Grid	Bad	Kalman filter	EKF	No
Gmapping SLAM	2D Lidar	Grid	Bad	Particle filter	Fast SLAM	Yes
Cartographer SLAM	2D Lidar/ IMU	Grid	Bad	Optimization based	Graph SLAM	Yes
RTAB-Map	RGB-D camera	Grid/Point cloud	Good	Optimization based	Graph SLAM	No
ORB-SLAM 3	Stereo camera	Trajectory	Good	Optimization based	Graph SLAM	No

Table [1]: Comparison table for SLAM algorithms

Table [1] summarizes the SLAM algorithms used with our dataset. Two SLAMs used the lidar, another two SLAMs used the cameras, and the other SLAM used both the lidar and IMU. They generated several types of maps depending on what sensor was used. The lidar SLAMs did not demonstrate satisfactory performance. Hector and Gmapping SLAM did not use any odometry data from the sensors. They estimated poses using the scan matching algorithms which were comparing and registering two consecutive laser scans. The issue was when the cart was moving in the straight and nondistinctive tunnels, the SLAM algorithms could not tell if the cart was moving forward or not. On the other hand, Cartographer SLAM used the IMU data to improve the pose estimation of the cart. It recognized the forward movements in the long tunnels better than the last two lidar SLAMs. However, it reacted too sensitively to the rotational movements.

The visual SLAMs achieved good performance. RTAB-Map was estimating accurate poses of the RGB-D camera based on the consecutive RGB-D images and its point cloud was describing the environment accurately too. If the camera scanned the tunnels thoroughly, the point cloud would be denser. Additionally, even though it experienced some drifts and deviations, the visual and loop closures corrected these errors and demonstrated their robustness. The other visual SLAM algorithm, ORB-SLAM 3 gave a good trajectory and it mostly matched the path that was taken to collect the data. It had some loop closure issues which resulted in around 3% error. To overcome this issue, rigorous calibration of the camera and data collection in open areas should be done.

Team member contributions

Each team member studied one or two SLAM algorithms and installed relevant ROS packages on their Ubuntu. The rosbag data were used with the SLAM algorithms individually and the results were shared throughout the team. Then, the report was divided into each SLAM and the other parts were finished altogether.

References

[1] "hector_slam - ROS Wiki," Ros.org. [Online]. Available: http://wiki.ros.org/hector_slam. [Accessed: 29-Apr-2023].

[2] "gmapping - ROS Wiki," Ros.org. [Online]. Available: http://wiki.ros.org/gmapping. [Accessed: 29-Apr-2023].

[3] "laser_scan_matcher - ROS Wiki," Ros.org. [Online]. Available: http://wiki.ros.org/laser_scan_matcher. [Accessed: 29-Apr-2023].

[4] "Cartographer ROS integration — cartographer ROS documentation," Readthedocs.io. [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/. [Accessed: 29-Apr-2023].

[5] "rtabmap_ros - ROS Wiki," Ros.org. [Online]. Available: http://wiki.ros.org/rtabmap_ros. [Accessed: 29-Apr-2023].

[6] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, 2011.

[7] W. Hess, D. Kohler, H. Rapp, and D. Andor, Real-Time Loop Closure in 2D LIDAR SLAM, in Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016.

[8] ORB_SLAM3: ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM.