Examining LeGO-LOAM

Jack Bauer, Sameer Bhatti, Riccardo Dalla, Afridi Shaik, Kevin Sinaga
Department of Electrical and Computer Engineering, Northeastern University

***Abstract*—This project aimed to utilize open sourced LeGO-LOAM packages to develop a further understanding of Lidar Odometry And Mapping (LOAM) in addition to testing the robustness of the package. We implemented LeGO-LOAM on a previously tested dataset, as provided by the group that published the package at the Steven's Institute of Technology, as well as on an untested dataset obtained from the Field Robotics Lab at Northeastern University.**

Implementation of this Ground Optimized version of LOAM allowed us to create a 3D map of the Steven's Institute of Technology dataset and the car/lidar dataset obtained by Northeastern's NUANCE car. LeGO-LOAM showed not to be as effective in producing loop closure in the Northeastern dataset while a variant, SC-LeGO-LOAM, had greater results in detecting loop closure and forming the map.

I. Introduction

Simultaneous Localization and Mapping, otherwise known as SLAM, is the computational problem of constructing a map of a previously unknown area while simultaneously keeping track of the location of the mapping device. One of the methods of SLAM is LOAM, or Lidar Odometry and Mapping, in which a 2-axis Lidar was used to generate a map and keep track of the movement of the mapping device. This project examines a further development of the LOAM algorithm called LeGO-LOAM, or Lightweight and Optimized Lidar Odometry and Mapping. LeGO-LOAM is said to be able to achieve real-time pose estimation on low-powered embedded systems.

The remainder of this report is organized as follows. Section II describes the algorithms tested during this project. Section III presents some of the issues that arised when running and testing the ROS packages used. Section IV shows the results of our experiments.

II. Hardware and Datasets

All of the datasets processed by the LeGO-LOAM packages were obtained using the VLP-16 Velodyne LiDAR sensor mounted upon various vehicles. The original Stevens Institute of Technology dataset was collected using a VLP-16 sensor mounted on a UVG called Clearpath Jackal while Northeastern's dataset was collected with the NUANCE car which has two VLP-16 sensors mounted on the roof. The VLP-16 utilizes 16 individual Lidars with a 30 degree pitch range paired with a 360 degree horizontal rotation. With a range of 100 meters and a rotation rate of 5 to 20 Hz, the VLP-16 can generate a surrounding point cloud consisting of 1800 by 16 data-points every second.

III. Methods

This section describes the algorithms involved in our investigation and testing on the LiDAR datasets.

A. LOAM

LOAM, or Lidar Odometry and Mapping, [1] aims to generate a 3-D map with low drift odometry using 2-axis Lidar moving in 6 degrees of freedom in real time. LOAM breaks down the problem of simultaneous localization and mapping, or SLAM, into two key algorithms. One algorithm estimates the velocity of the Lidar at high frequency but with low fidelity by way of fast computations, while the other algorithm conducts fine matching and point cloud registration by examining geometric distributions of local point clusters running at a frequency much lower than the first. In essence, both algorithms work to extract extract feature points on sharp edges, then and match them to edge line segments and planar surface patches, in order to estimate motion for odometry as well as to generate maps.

The algorithm running at high frequency ensures that a preliminary motion estimate can be obtained as the higher precision map is being computed. To perform motion estimation, transform and rotation matrices are obtained via Levenberg-Marquadt (LM) optimization to match all the planar and edge features extracted from the current scan to those from the previous scan. The mapping algorithm functions by way of batch optimization, which works in a similar manner to Iterative Closest Point (ICP). With the two algorithms running in parallel, LOAM solves the SLAM problem in real time.

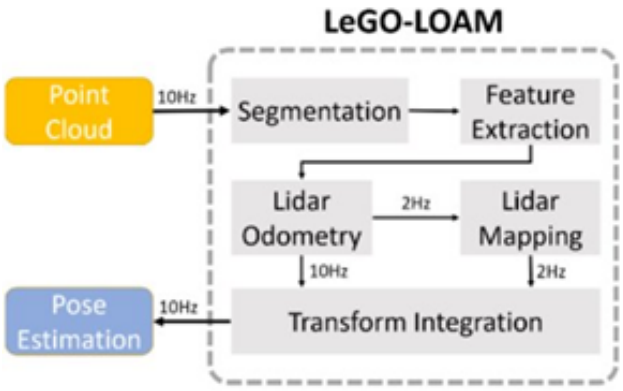

Fig. 1. LeGO-LOAM flowchart

B. LeGO-LOAM

LeGO-LOAM stands for Lightweight and Ground Optimized LOAM [2]. In fact, this algorithm expands LOAM to introduce a few substantial improvements to the extraction and matching steps. An initial segmentation is applied to the point

cloud to discard points that might represent unreliable features The cloud is divided into clusters and only the clusters wit more than 30 points make it to the feature extraction step. Th points are also marked as ground points if they are thought t come from the ground. This step is useful to segment out al the points that don't come from clear and big objects in th scene and might be noise. Then, edge and surface features ar extracted based on their roughness value. The pose estimate o the sensor is calculated by fusion of the estimate found by th Odometry and Mapping nodes. As is the case with LOAM the Lidar Odometry node runs at 10Hz and provides a fast but somewhat inaccurate pose estimate. It does so by a applying two step, i.e. edge features first and then planar features, Levenberg-Marquardt optimization to match the current scan to the previous one. The Lidar Mapping node runs instead at only 2Hz as it is responsible for matching the current scan to the set of features from the previous 100 scans. It then stores the newly matched features in the map and computes a more accurate pose of the sensor. Scan matching is achieved in different ways depending on the package. The ROS packages tested use ICP and scan context, and yield different results as presented in the following sections.

a) Iterative Closet Point: Iterative Closest Point(ICP) is an algorithm that calculates the best rotation matrix **R** and a translation vector **t** between two sets of point clouds **P** = $\{p_1, ..., p_M\}$ and **Q** = $\{q_1, ..., q_N\}$. The objective is to optimize a rigid transformation on **P** using rotation matrix **R** and translation vector **t** to align **P** and **Q**.

$$min_{R,t} \sum_{i=1}^{M} D_i(\mathbf{R}, \mathbf{t})^2 + I_{SO}(d)(\mathbf{R}) \tag{1}$$

where $D_i(\mathbf{R}, \mathbf{t}) = min_q|\mathbf{R}p_i + \mathbf{t} - q|$ is the distance from the transformed point $\mathbf{R}p_i + t$ and points in the target point cloud Q and I_{SO} is an indicator function for special orthogonal group SO(d). The indicator function will take a value of 0 if $R^T R = 1$.

The ICP algorithm tries to solve this problem by alternating between correspondence and alignment steps iteratively. In the correspondence step, a closest point q_i in **Q** for each point in **P** based on the transformation (**R**, **t**) are calculated. In the alignment step, the transformation is updated by minimizing the distance between the corresponding points using Singular Vector Decomposition(SVD). For the better performance of the algorithm, R and t are initialized with the transformation calculated from the previous iteration.

The loop-closure of LeGO-LOAM uses point-to-point variant of the ICP algorithm. It often fails when the odometry drift is too large. To overcome the loop-closure detection failures, SC-LeGO-LOAM uses scan context to detect loop effectively.

b) Scan context: Scan Context [3] is a non histogram-based global descriptor for 3D Light Detection and Ranging (LiDAR) scans. To create a scan context of a a point cloud, the scan is divided into equally spaced azimuthal and radial bins in the sensor coordinate. For each bin, a single real value is assigned by using the cloud of points that are in the region

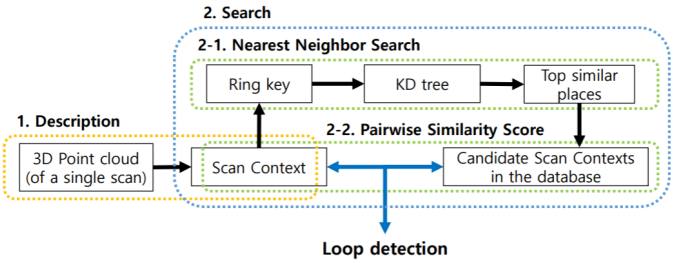

Fig. 2. Loop detection using the scan contexts of LiDAR scans

of the bin. One common approach is use to the biggest z-coordinate value from all the points. For the bins with no points, a value of zero will be allocated.

For the loop detection process, the current scan context will be compared to the top old scan contexts and if the closest scan context is found on the database then it is considered to be a loop. To determine if two scan contexts are closer, column-wise cosine distance metric is used and it is further normalized by the number of columns. The top scans contexts are filtered with the nearest neighbors search criteria on a KD-Tree data structure. Scan context and the two-phase search algorithm make the loop detection invariant to LiDAR viewpoint changes so that loops can be detected in places such as reverse revisit and corner.

IV. Issues

One of the biggest issues we encountered completing this project was the fact that multiple members of the group were not able to visualize the simulation properly. Even though we were all able to run LeGO-LOAM on all the datasets considered, the majority of the group was not able to see the full point clouds in Rviz. This is believed to be due to issues related to hardware constraints, the use of virtual machines and the inability to make use of the GPU when running ROS. However, at least one person in the group was able to see the simulation properly and shared the results in real time with the rest of the group.

Once we were able to run and visualize the results on the dataset provided by Northeastern University and recorded with the NUANCE autonomous car, we encountered a second issue. The LeGO-LOAM ROS package we used takes in LiDAR and IMU data by subscribing to the respective ROS topics. However, as will be shown in the following section, when we provided both datasets, the algorithm completely failed to construct a realistic trajectory and map of the environment. The reason for this became clear after a closer inspection of the instruction provided by the developers. The optional IMU data provided to the algorithm is used to correct the point cloud for distortion caused by sensor motion. However, is the IMU is not properly aligned with the LiDAR it can deteriorate the results, as happened in our case. For this reason, we decided to use only the LiDAR datasets for our project. It is worth noting

that in LeGO-LOAM BOR, IMU data is actually ignored and focuses exclusively on point cloud data. We attempted to run LeGO-Loam BOR on the data as well but resulted in several cmake errors related to structure definitions in the code. We tried to correct the code without erasing the effect of the code but were unsuccessful and could not produce a visualization of the data with LeGO-Loam BOR.

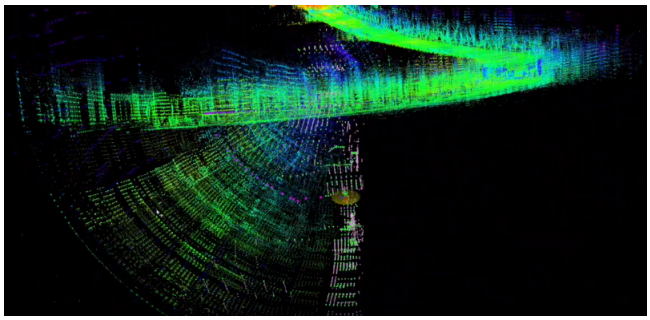

Fig. 3. Very inaccurate results by using IMU data on LeGO-Loam

V. Tests and Results

LeGO-Loam was run on both the dataset provided by Steven's Institute of Technology and the car/lidar NU dataset. LeGO-Loam was successful in finding the trajectory and mapping the environment in the Steven's dataset including having successful loop closure. However, in the NU dataset, LeGO-Loam failed to provide an accurate trajectory of the path taken in the car. The IMU data seemed to have caused the issue. When it was removed, the path matched the path obtained from GPS data, showing a more accurate trajectory. However, Lego-LOAM did not produce successful loop closure. It seems we were not the only ones with this issue since there was a separate group that commented the same problem on Github with LeGO-LOAM not properly producing loop closure, especially in the z axis. This seems to be attributed to the ICP algorithm used since it would not be able to align the previous maps with a drift in odometry.

Scan Context LeGO-Loam was used to test if that theory were true since it replaces the use of ICP. Using this without IMU data proved to be the better method of obtaining loop closure while also accurately mapping the environment. It is worth noting the change in altitude throughout mapping. It seems that there tends to be a drift in the z-axis while mapping. The actual altitude according to GPS data remained constant throughout the trip. In both LeGO-Loam and SC-LeGO-Loam, the z axis tended to drift upward.

VI. Conclusion

LeGO-Loam could seem like a good option to use if the map obtained were to remain in 2D. Perhaps if it were used to make a 2D map rather than a 3D map, LeGO-Loam could have better results. LeGO-Loam does not seem like the viable option in use for large datasets of larger area. Although it worked well for the Steven's dataset, it did not show successful loop closure in the car dataset where the odometry drift was larger.

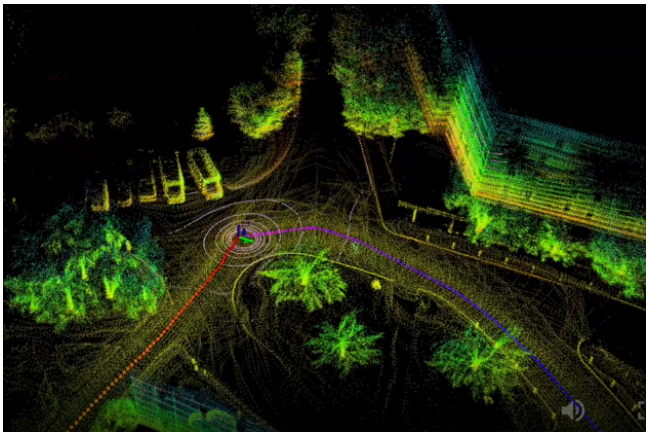

Fig. 4. LeGO-LOAM performing loop closure on SIT dataset

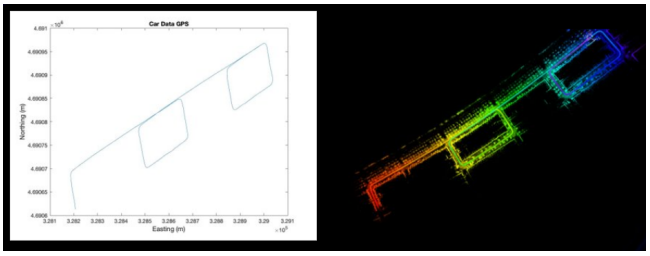

Fig. 5. SC-LeGO-LOAM on NU dataset vs GPS data

SC-LeGO-Loam seems to be the better option in producing accurate loop closure and trajectory mapping, despite drift in the z axis.

References

[1] Ji Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, 2014.

[2] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[3] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3D point cloud map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Oct. 2018.