Implementing LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping

Mobile Robotics Team 11
Bowen Mu, Chien Erh Lin, Haochen Wu, and Weilin Xu
University of Michigan
Ann Arbor, USA
Email: mubowen@umich.edu; chienerh@umich.edu; haochenw@umich.edu; xuweilin@umich.edu

***Abstract*—LeGO-LOAM, a lightweight and ground-optimized lidar odometry and mapping method, is able to do six degree-of-freedom pose estimation for real time with ground vehicles [4]. This project re-implemented LeGO-LOAM which reduced computational expense while keeping similar accuracy compared to LOAM method. The whole LeGO-LOAM system is implemented in Robot Operating System (ROS). The data used for simulation is raw data (with Velodyne LiDAR data and IMU) from KITTI odometry benchmark dataset. Sequences 00, 05, and 08 are tested in our project. The mapping and odometry results of these sequences are presented in this report. Our code is public in the GitHub Repository. Video of implementing LeGO-LOAM on Kitti sequence 00 and sequence 08 are available at youtu.be/z3yXCRAMC18, and youtu.be/nJ74Uk6m10U.**

I. Introduction

Great efforts have been made to achieve real-time 6 degree-of-freedom simultaneous localization and mapping (SLAM). Both visual-based methods and LiDAR-based methods are widely used in this field. Compared to visual-based methods, LiDAR-based methods can function independent of the light intensity and provide more detailed information with a higher resolution. In addition, mapping with LiDARs can provide high frequency range measurements where errors are relatively constant irrespective of the distances measured.

Feature-based matching methods are more common because they require less computing resources by extracting representative features from the environment. Point Feature Histograms (PFH) [2] and Viewpoint Feature Histograms (VFH) [3], are some of the feature-detect methods that have been proposed for extracting features from point clouds.

LOAM, a low-drift and real-time lidar odometry and mapping method, is proposed in [5] and [6]. LOAM achieves both low-drift and low-computational complexity in real-time. It can do mapping in real time because it matches and registers the point cloud at a lower frequency while performing odometry at a high frequency. LOAM performs point feature to edge/plane scan-matching to find correspondences between scans. Features are extracted by calculating the roughness of a point in its local region. The points with high and low roughness values are selected as edge features and planar features respectively.

In this project, we re-implement LeGO-LOAM: lightweight and ground-optimized lidar odometry and mapping method.

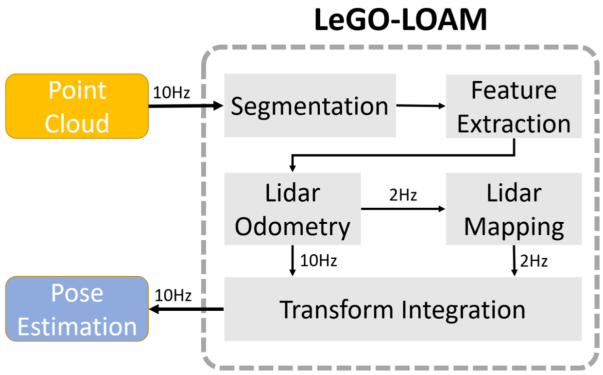


Fig. 1: LeGO-LOAM system overview. First apply point cloud segmentation to filter out noise. Then do feature extraction to obtain distinctive planar and edge features. Lidar Odometry estimates the sensor motion between two scans. Lidar Mapping algorithm takes the projected point cloud from lidar odometry, And transforms the point cloud into map coordinates.

The system overview of LeGO-LOAM is shown in Fig. 1. We first use a single scan point cloud for segmentation and then project it onto the range image for segmentation. Then, we use the features extracted from the previous module to find the transformation associated with the last scan. These features are further processed in the LiDAR map and registered to the global point cloud map. Finally, the transform integral module combines the attitude estimation results of lidar odometry and LiDAR mapping, and outputs the final pose estimate.

II. Methodology

A. LIDAR Segmentation Step

LiDAR segmentation step improves processing efficiency and feature extraction accuracy. With image projection algorithm and point cloud segmentation method, small objects in a noisy environment is filtered out and ground points are segmented.

1) Image Projection: Point Cloud is first projected onto the range image. The resolution of the projected distance image comes from the horizontal and vertical angular resolution of the LiDAR device. Each valid point in the point cloud is now

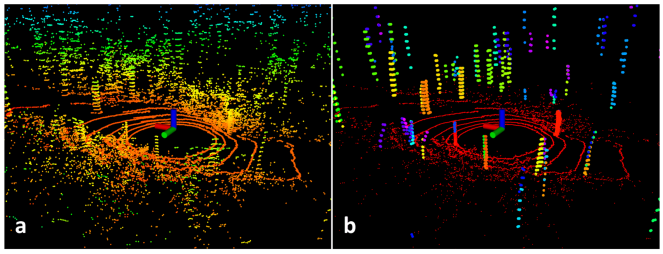


Fig. 2: Visualization of (a) original point cloud and (b) result of point cloud segmentation. The red points are ground points. The other points are other segmented clusters. This segmentation step filtered out small objects in the noisy environment.

represented by a unique pixel in the range image, and the value is the Euclidean distance. A ground plane estimate is performed prior to segmentation. After this process, points that may represent the ground are marked as ground points and not used for segmentation.

2) Point Cloud Segmentation: Point cloud segmentation is performed to discard points that may represent unreliable features after ground separation. This split method is applied to range images to group points into multiple clusters and assign a unique label to each cluster. In order to perform fast and reliable feature extraction using segmented point clouds, we omit clusters of less than 30 points. Applying segmentation to a point cloud can improve processing efficiency and feature extraction accuracy. The visualization of point cloud segmentation is shown in Fig. 2.

B. Algorithm for feature Association

The current position of the robot $(z, roll, pitch)$ could be localized from the ground feature. Similarly to the LOAM[4], we will match the point cloud in each scan to update the position of robot (x, y, yaw).

In this algorithm, we will use GTSAM C++ base which is used for smoothing and mapping in the robotic and computer vision area. Comparing with the g2o (A General Framework for Graph Optimization) which uses sparse matrix to solve nonlinear optimization problems, GTSAM would use factor graphs and Bayes networks to maximize the posterior probability.

There are two parts of this algorithm: 1) Feature point extraction and 2) Feature point matching.

1) Feature point extraction: For the most point cloud $\mathcal{P}_k$ extracted from LiDARs, many 3D LiDARs naturally generate unevenly distributed points in $\mathcal{P}_k$. For this project, we choose the LiDAR with the horizontal and vertical angular resolution of $0.2°$ and $2°$ respectively. The resolution of the projected range image is 1800 by 16. To improve the efficiency and reduce the complicity of calculation, instead of extracting features from raw point clouds, we extract features from ground points and segmented points. Let S be the set of continuous points of p_i from the same row of the range image. Half of the points in S are on either side of p_i. For this project, the magnitude of S would be set as 10. Using the range values

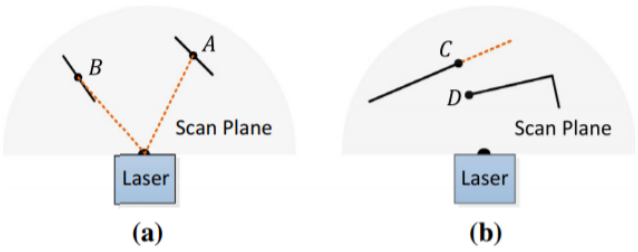


Fig. 3: a) The solid line segments represent local surface patches. The point A is on a surface that has an angle to the laser beam (the orange dot line segment). The point B is on a surface patch that roughly parallel to the laser beam. We decide to treat point B as unreliable laser and do not select since it does not have enough feature information. b) The solid line segments are observable objects by the laser. The point C is on the boundary of an occluded region which is the orange dot line segment. It could be detected as an edge point. However, if LiDAR scan at a different angle, the occluded region can change and be observable. Therefore, we don't choose point D as a salient edge point or select it as a feature point

computed during segmentation, we can evaluate the roughness of point p_i is S,

$$c = \frac{1}{|S| \cdot \|r_i\|} \| \sum_{j \in S, j \neq i} (r_j - r_i) \| \tag{1}$$

In order to evenly extract features from all directions, we will divide the range image horizontally into several equal sub-image based on their roughness c. We will use a threshold c_{th} to distinguish different type of features. The points in a scan are sorted based on the c values, then feature points are selected with the c larger than c_{th} called edge points, and the c smaller then c_{th} called planar features points. Then n_{F_c} edge feature points with maximum c, which do not belong to the ground, are selected from each row in the sub-image. n_{F_p} planar feature points with the minimum c, which may be labeled as either ground or segmented points, are selected in the same way, such as figure 3. From all sub-images, we define F_e and F_p to be the set of all edges and planar features from all sub-images. We then extract n_{F_e} edge features with the maximum c, which does not belong to the ground, such as figure 4.

From each row in the sub-image. Similarly, we extract n_{F_p} planar features with the minimum c, which must be ground points, from each row in the sub-image. Let F_e and F_p be the set of all edge and planar features from this process. For this project, we divide the 360-degree range image into 6 sub-images. Each sub-image has a resolution of 300 by 16.

When selecting the feature point, we need to choose the best point. we need to avoid points whose surrounded points are selected or points on local planar surfaces that are roughly parallel to the laser beams. From Fig.3a, the solid line A and B represent the local planar surface that is parallel to each other. We consider these points are normally unreliable. In

addition, we also want to avoid the points which are on the boundary of occluded regions (such as Fig.3b). In Fig.3b, point C is an edge point on the LiDAR point. The dashed segment represents the blocked connected surface by another object. If LiDAR moves to another direction, the occluded region may change and become observable. In order to avoid selecting the unreliable points, we find again the set of points S. We define the point i can be selected only if S does not form a surface patch whose normal is within $10°$ to the laser beam, and there is no point in S that is disconnected from i by a gap in the direction of the laser beam and is at the same time closer to the LiDAR then point i such as point B in Fig.3b.

In the summary, we would choose the feature points as edge points starting from the maximum c value, and the feature points as planar points starting from the minimum c value. There are three standards for us to determine the reliability of the points, as follows:

- The number of selected edge points or planar points cannot exceed the maximum of the sub region
- None of its surrounding point is already selected
- It cannot be on a surface patch whose normal is within $10°$ to the laser beam, or on boundary of an occluded region.

2) Finding feature point correspondence: The odometry algorithm estimates motion of the LiDAR within a sweep. For the time t_k to be the starting time of a sweep k. At the end of sweep k-1, the point cloud perceived during the sweep, $\mathcal{P}_{k-1}$, is projected to time stamp t_k which illustrated in the figure 5. We define the projected point cloud as $\overline{\mathcal{P}}_{k-1}$. During the next sweep k, $\overline{\mathcal{P}}_{k-1}$ is used together with the newly received point cloud, $\mathcal{P}_{k-1}$, to estimate the motion of the LiDAR.

III. LeGO LiDAR Odometry Algorithm

The LiDAR odometry module estimates the sensor motion between the two consecutive scans. The transformation between two scans is found by performing point-to-edge and point-to-plane scan-matching. We need to find the corresponding features for the points in the current point cloud and previous point clouds.

In order to improve feature matching efficiency and matching accuracy. In the project, we will make some improvement

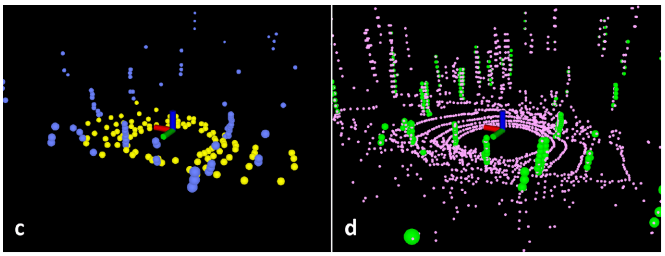


Fig. 4: From the plot (d), the pink point and green point represent the edge and planar feature with out filtering. In (c), the blue point and yellow point represent the number of sharpest and flattest points of edge and planar feature in F_e and F_p

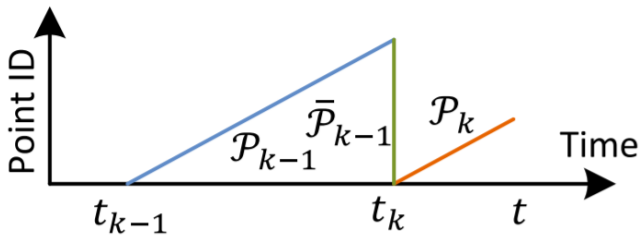


Fig. 5: Project point cloud to the end of a sweep. The blue colored line segment represents the point cloud perceived during sweep k, $\mathcal{P}_{k-1}$. At the end of sweep k-1, $\mathcal{P}_{k-1}$ is projected to time stamp t_k to obtain $\overline{\mathcal{P}}_{k-1}$ (the green colored line segment). Then, during sweep k, $\overline{\mathcal{P}}_{k-1}$ and the newly perceived point cloud $\mathcal{P}_k$ (the orange colored line segment) are used together to estimate the lidar motion (Color figure online)

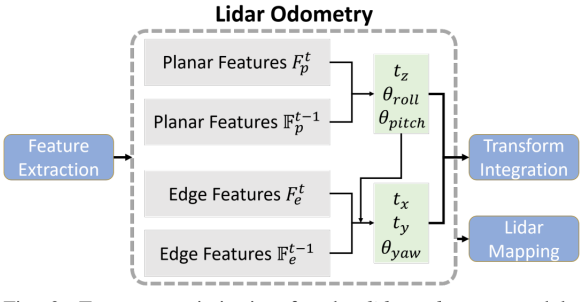


Fig. 6: Two-step optimization for LeGO LiDAR odometry. $[t_z, \theta_{roll}, \theta_{pitch}]$ is first obtained and optimized by matching the planar features extracted from ground points. $[t_x, t_y, \theta_{yaw}]$ are then estimated using the extracted edge features while considering other states as constraints.

for the LeGO-LOAM based on LOAM algorithm as shown in Algorithm 1.

1) Label Matching: Since each feature point is labeled after segmentation, we could only find correspondences that have the same label from the points from the previous scan. For planar feature points, only points that are labeled as ground points in the previous point cloud are used for computing correspondence distances. For edge feature points, the corresponding edge line points are found from the previously segmented clusters. This label matching method would improve the matching accuracy by only considering the points with the same category to prevent crossover mistakes. This process could filter out the outliers and accurately locate the potential candidates for computing correspondence distance.

2) Two-step L-M Optimization: In LOAM, the distances between the edge and planar feature points from the current scan and the corresponding feature points from the previous scan are computed respectively as equation (2) and (3). $\overline{X}$ denotes the edge or planar feature points in one scan. $\widetilde{X}$ denotes the edge or planar feature points in the consecutive

Algorithm 1 LeGO Feature Extraction and Lidar Odometry

input: Segmented Point Clouds from last scan $\overline{P}k$ with EDGE and PLANAR labels and from current scan P_{k+1}; Pose transformation T_{k+1} P
output: $\overline{P}_{k+1}$; optimized T_{k+1}
if at the beginning of the sweep **then**
  $T_{k+1} \leftarrow 0$
end if
Find edge and planar feature points in P_{k+1} and store as ε_{k+1} and $\mathcal{H}_{k+1}$
for $i \leq MaxIteration$ **do**
  for $\forall$ point $\in \mathcal{H}_{k+1}$ **do**
    find correspondences with PLANAR labelled points and compute distances as equation (3)
  end for
  Update $[t_z, \theta_{roll}, \theta_{pitch}]$ by performing L-M Optimization
  for $\forall$ point $\in \varepsilon_{k+1}$ **do**
    find correspondences with EDGE labelled points and compute distances as equation (2)
  end for
  Update $[t_x, t_y, \theta_{yaw}]$ by performing L-M Optimization with $[t_z, \theta_{roll}, \theta_{pitch}]$ as constraints, get T_{k+1}.
  if at the end of the sweep **then**
    Project P_{k+1} and get $\overline{P}_{k+1}$
    return $T_{k+1}, \overline{P}_{k+1}$
  else
    return T_{k+1}
  end if
end for

scan. (j, l) describes the correspondences for edge feature points, and (j, l, m) describes the correspondences for planar feature points. Then the distances for the edge and planar feature points are compiled into a single distance vector as a nonlinear equation (4), which could be optimized by the Levenberg-Marquardt method as equation (5) by minimizing d toward zero, where J is the Jacobian matrix of the nonlinear function f.

$$d_\varepsilon = \frac{|(\widetilde{X}_{k+1,i} - \overline{X}_{k,j}) \times (\widetilde{X}_{k+1,i} - \overline{X}_{k,l})|}{|\widetilde{X}_{k,j} - \overline{X}_{k,l}|} \tag{2}$$

$$d_\mathcal{H} = \frac{|(\widetilde{X}_{k+1,i} - \overline{X}_{k,j})(\overline{X}_{k,j} - \overline{X}_{k,l}) \times (\overline{X}_{k,j} - \overline{X}_{k,m})|}{|(\overline{X}_{k,j} - \overline{X}_{k,l}) \times (\overline{X}_{k,j} - \overline{X}_{k,m})|} \tag{3}$$

$$f(T_{k+1}) = d \tag{4}$$

$$T_{k+1} = T_{k+1} - (J^T J + \lambda diag(J^T J))^{-1} J^T d \tag{5}$$

However, this method is computationally intensive because all six states are considered together. LeGO-LOAM proposed a two-step L-M Optimization as shown in Figure 6. The optimal pose transformation T is found in two steps: First, $[t_z, \theta_{roll}, \theta_{pitch}]$ are estimated by matching the planar features and their correspondences in the previous scan. Second, $[t_x, t_y, \theta_{yaw}]$ are then estimated using the edge features and their correspondences in the previous scan while applying $[t_z, \theta_{roll}, \theta_{pitch}]$ as constraints. Therefore, LeGO-LOAM reduces the 6D transformation optimization process to two steps of 3D optimization process, which is much more efficient than LOAM and similar accuracy can be maintained.

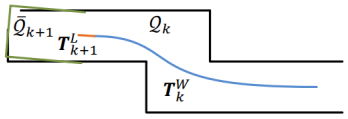


Fig. 7: Mapping Process

A. LeGO-LiDAR Mapping Algorithm

1) Mapping: It can be recalled that the LiDAR mapping algorithm runs at a lower frequency than the Lidar odometry. LiDAR mapping algorithm take the outputs of Lidar odometry (LiDAR pose transformation T_{k+1}, and reprojected the point cloud $\overline{P}_{k+1}$) at the end of sweep $k+1$. The mapping algorithm then matches $\overline{P}_{k+1}$ from Lidar coordinates to world coordinates, as shown in Figure 7. $\mathcal{Q}_k$ are the current point cloud on the map, T^W_k is the Lidar pose transformation and the end of sweep k. The mapping algorithm extends T^W_k for one more step T^W_{k+1} as shown in Figure 8.

To extract the feature points on the map, the process is the same as Section II-B1. To find the correspondences between the current map cloud and $\overline{Q}_{k+1}$, instead of those points in a 10m cube region in Q_k that intersect with $\overline{Q}_{k+1}$ being extracted in LOAM, points in a 100m cube region in the history of key frame point clouds are extracted. A covariance matrix, and its eigenvalues and eigenvectors of surrounding points could be computed. If the points are distributed on an edge line, one eigenvalue would be significantly greater than others; if the points are distributed on a planar patch, one eigenvalue would be significantly smaller than others.

2) Loop Closure: Another main difference between LeGO-LOAM and LOAM is that it has the ability to detect loop closure. Loop closure detection runs at a lower frequency and is feature-pose based. Since the history of the keyframe point clouds is stored, it grants the algorithm to perform loop closure and better mapping. The detection returns true if two point cloud frames are separated by time duration, within the searching region, and edge and planar feature points are close to each other.

3) Transform Integration: Integration of the pose transforms is demonstrated in Figure 8. The blue region repre-

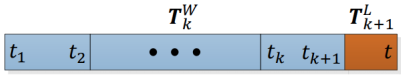


Fig. 8: Illustration of Transformation Integration

sents the pose transformation output from LiDAR mapping algorithm generated once per sweep at a lower frequency. The orange region represents the transform output from the Lidar odometry algorithm at a higher frequency. The LiDAR pose transformation in the world coordinates is computed by combining two transformations at the same frequency as the LiDAR odometry.

IV. Results

A. Implementation

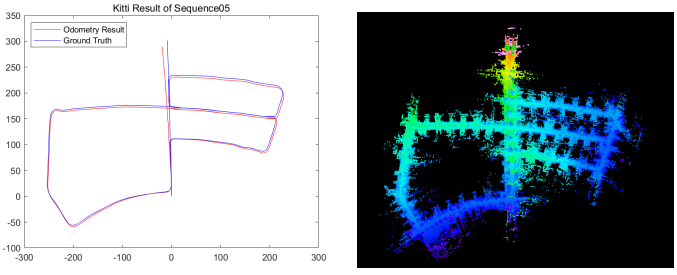
The whole LeGO-LOAM system is implemented in Robot Operating System (ROS). The data used for simulation is raw data (with Velodyne LiDAR data and IMU) from KITTI dataset [1]. The structure of the system is illustrated in Fig. 12.

The ROSNode *ImageProjection* is for segmentation and feature extraction functions, the ROSNode *FeatureAssoication* for LiDAR odometry function, the ROSNode *MapOptimization* for Lidar mapping, and *TransformFusion* node for transform integration. The communication between ROSNodes are realized by ROSTopic: */kitti/velo/pointcloud* is the Lidar input, */segmented cloud* is the segmented pointcloud data with edge and ground plane information from feature extraction, */laser odmo to init* is the odometry result of LiDAR odometry, */aft mapped to init* is the odometry result of LiDAR mapping, and */integrated to init* is the final output for odometry after transform integration. All the four core nodes are composed using C++.

B. Odometry Result

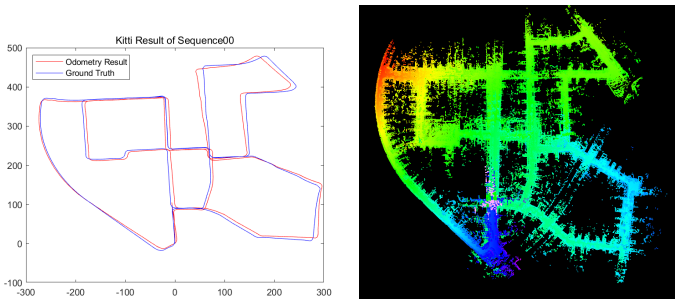
To evaluate the performance of odometry, the predicted pose published by *TransformFusion* are printed in the terminal and saved in .txt files, while the ground truth data are provided on KITTI dataset.

Then, the odometry results for kitti data of sequence 05, 00, and 08 are shown in Fig. 9, 10 and 11, respectively. The figures and the error calculation are operated in a MATLAB file. As the figures show, the odemetry result is very similar to the ground truth, with only a small drift. Video of implementing LeGO-LOAM on Kitti sequence 00 and sequence 08 are available at youtu.be/z3yXCRAMC18, and youtu.be/nJ74Uk6m10U.



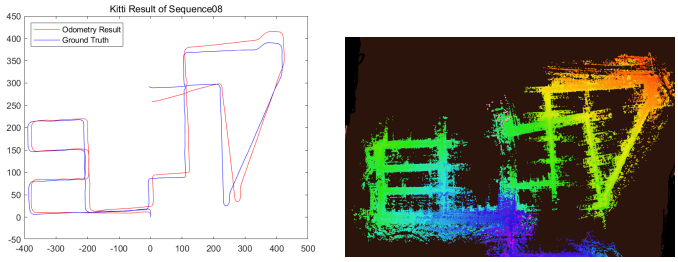
(a) (b)

Fig. 9: (a) Comparison between simulation result and ground truth and (b) mapping result for Sequence 05 of Kitti are shown.



(a) (b)

Fig. 10: (a) Comparison between simulation result and ground truth and (b) mapping result for Sequence 00 of Kitti are shown. Loop closure property is observed at the trajectory between around (0m, 250m) and (0m and 350m)



(a) (b)

Fig. 11: (a) Comparison between simulation result and ground truth and (b) mapping result for Sequence 08 of Kitti are shown.

To evaluate the odometry result, the position error is defined in Eq. 6:

$$e_t = \sqrt{(x_t - x_t^{GT})^2 + (y_t - y_t^{GT})^2 + (z_t - z_t^{GT})^2} \quad (6)$$

where (x_t, y_t, z_t) and $(x_t^{GT}, y_t^{GT}, z_t^{GT})$ are the simulation and ground truth pose at time t. The position error for Sequence 05 and 00 are shown in Fig. 13 (a) (c), respectively. The average position error is 6.41 m, 7.35 m and 18.68 m for Sequence 05, 00 and 08, respectively, which is quite satisfying when compared with the field size of around 500 m $\times$ 500m.

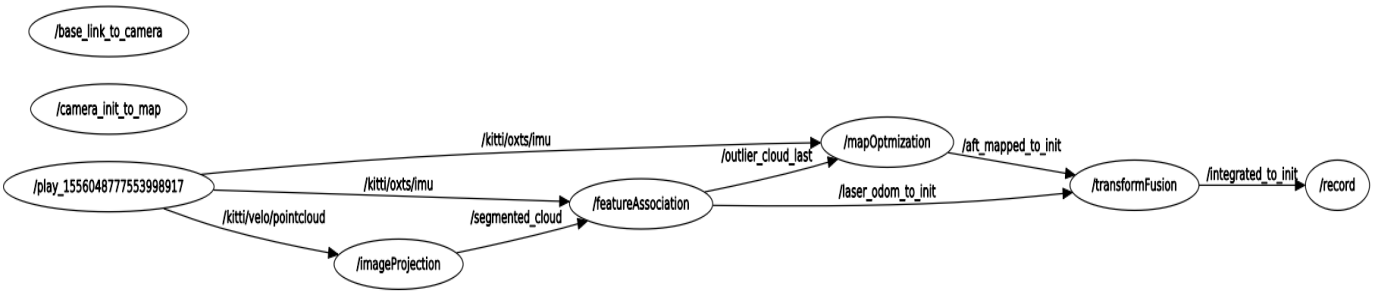
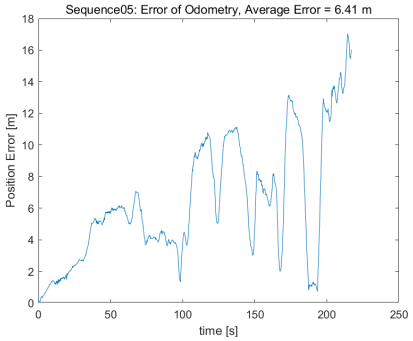
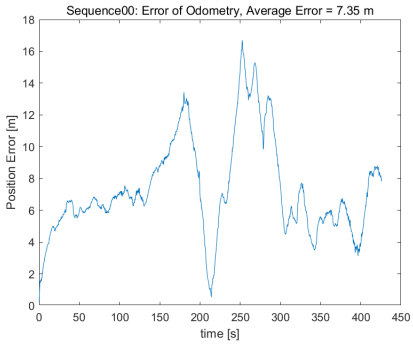


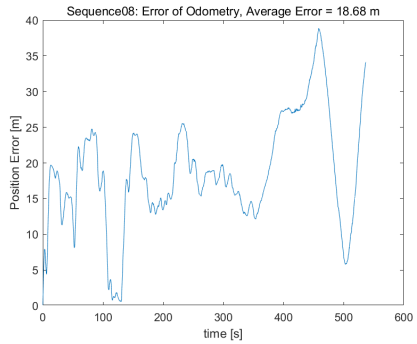
Fig. 12: Illustration of Transformation Integration in ROS is shown. The LeGO-LOAM system includes node of *ImageProjection*, *FeatureAssoication*, *MapOptimization* and *TransformFusion*The velodyne data from KITTI dataset is the input of the system, and the odometry (6D pose) is the final output.



(a)



(b)



(c)

Fig. 13: Odometry position error for (a) Sequence 05, (b) Sequence 00 and (c) Sequence 08 of KITTI is shown. In (a), the error increases at each turning corner in the trajectory. In (b), the error drops significantly at time around $t = 220$s since there is a loop closure. (c) has the highest error since it has more and sharper corners in the trajectory.

The position error for Sequence 00 is larger than that of Sequence05. It agrees with the theory since its reason is that Sequence 00 has a more sharp turn. The sharp turn breaks the assumption that there is no drift over a short period of time for the transform integration step. In Sequence 08, there are even sharper and more corners in the trajectory, and, thus, the position error is even much higher than that of Sequence 00.

In addition, for Sequence 00, the position error significantly decreases at time 220s, which corresponds to the road between (0m, 250m) and (0m, 350m). This is a loop closure, so it is reasonable that the odometry error drops significantly.

C. Mapping Result

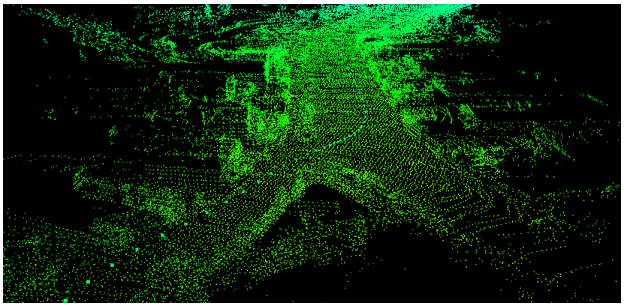
The mapping results for kitti data of sequence 05 and 00 are shown in Fig. 9(b) and 10(b), respectively. The darker the color is, the altitude (z) is smaller, which has been verified by the ground truth data. The overall mapping quality is satisfying. For example, Fig. 14 shows a comparison of a scene in the data set. While the mapping result clearly shows the outline of the Y-type crossroad and the surrounding building, while details of objects are lost. It is acceptable because it is a trade-off between the efficiency of the algorithm and details of mapping.

V. Challenges

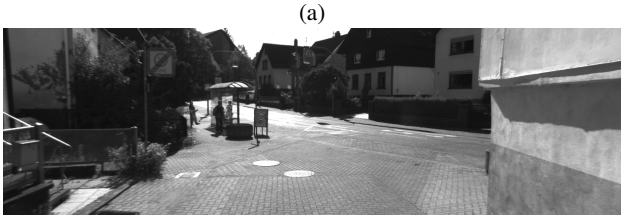
While re-implementing the algorithm, we found there is a few challenges exist for the Lego-algorithm.

The first challenge occurs when we run the KITTI data set. When the vehicle drives at crossroads, the algorithm would return a wrong transformation that there is a shift exist for the mapping. This behavior appears at sharp turns also. We believe this happened because of the wrong transformations that if IMU data is not aligned with the Velodyne axis. After the test, for the data set without using IMU, it indeed performs much less drift but still unable to eliminate the drift. In the next step, the team would try to re-calibrate the IMU data and try to solve this issue.

Another challenge is that although LeGO-LOAM is a high-efficiency algorithm, it only has 10 Hz for the point cloud and pose estimation. Considering that the data within one minute



(a)



(b)

Fig. 14: (a) Mapping result (b) camera data for a scene in KITTI data are shown. The mapping result clearly shows the outline of the Y-type crossing and the surrounding building, while details of objects are lost.

is around 4 GB, the algorithm is of high demanding for the processor when implemented on board. This issue would limit the reliability of a single scan and may lose information for a fast speed vehicle.

Furthermore, the algorithm does not distinguish between moving and static objects. When the features on a moving object are taken for localization, the performance of the algorithm is negatively affected. It is part of the reason for the error in odometry estimation. It also explains why the loop closure does not improve the odometry estimation sometimes.

In addition, understanding and debugging for the whole system framework (the functions of each rosnode and rostopic) in ROS is challenging for us. The delay and asynchronous communication between nodes affected the performance of the algorithm. For example, it is observed that the simulation result for the same data input is not completely identical on different computers and different trials.

VI. Conclusion

LeGO-LOAM efficiently and accurately utilizes LiDAR point cloud data for SLAM problem. It takes point cloud data as input and outputs the 6D pose for odometry and mapping. The algorithm is re-implemented by us in a pre-built ROS frame. The results of simulation verified the performance of LeGO-LOAM and are consistent with the theory of LeGO-LOAM.

LeGO-LOAM involves segmentation, feature extraction, LiDAR odometry, LiDAR mapping, and transformation integration steps. Compared with typical LOAM algorithm, LeGO-LOAM is more computationally efficient. It extracts two kinds of features, plane, and edge, based the criterion of smoothness from point cloud data, and significantly reduces the amount of data for the association. The computation for matching of selected feature points is further reduced since the matching is operated only with groups of features with the same label. Then, a two-step L-M optimization efficiently estimates $(t_x, \theta_{roll}, \theta_{pitch})$ from plane features first, and (t_y, t_z, θ_{yaw}) from edge features next for LiDAR odometry. Finally, mapping is operated at a lower frequency and integrated with odometry result. In addition, the ability to detect loop closure enables LeGO-LOAM to do long distance mapping and odometry.

The simulation result on KITTI data set illustrates that position estimation error for odometry is acceptable (around 10 m) and that mapping result can provide concise but adequate information of the environment. It is also observed that the loop closure reduces the error in odometry. However, when the drift is large within a short time (such as sharp turn), the performance of LeGO-LOAM is negatively affected.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[2] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 643–650. IEEE, 2008.

[3] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162. IEEE, 2010.

[4] 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.

[5] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9, 2014.

[6] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.