LO-Net: Deep Real-time Lidar Odometry

Qing Li\(^1\), Shaoyang Chen\(^1\), Cheng Wang\(^1\), Xin Li\(^2\), Chenglu Wen\(^1\), Ming Cheng\(^1\), Jonathan Li\(^1\)
\(^1\)Xiamen University, Fujian, China
\(^2\)Louisiana State University, Louisiana, USA
{hello.qingli,tinyyoh}@gmail.com, xinli@cct.lsu.ed, {cwang,clwen,chw99,junli}@xmu.edu.cn

Abstract

We present a novel deep convolutional network pipeline, LO-Net, for real-time lidar odometry estimation. Unlike most existing lidar odometry (LO) estimations that go through individually designed feature selection, feature matching, and pose estimation pipeline, LO-Net can be trained in an end-to-end manner. With a new mask-weighted geometric constraint loss, LO-Net can effectively learn feature representation for LO estimation, and can implicitly exploit the sequential dependencies and dynamics in the data. We also design a scan-to-map module, which uses the geometric and semantic information learned in LO-Net, to improve the estimation accuracy. Experiments on benchmark datasets demonstrate that LO-Net outperforms existing learning based approaches and has similar accuracy with the state-of-the-art geometry-based approach, LOAM.

1. Introduction

Estimating 3D position and orientation of a mobile platform is a fundamental problem in 3D computer vision, and it provides important navigation information for robotics and autonomous driving. Mobile platforms usually collect information from the real-time perception of the environment and use on-board sensors such as lidars, Inertial Measurement Units (IMU), or cameras, to estimate their motions. Lidar can obtain robust features of different environments as it is not sensitive to lighting conditions, and it also acquires more accurate distance information than cameras. Therefore, developing an accurate and robust real-time lidar odometry estimation system is desirable.

Classic lidar-based registration methods used in pose estimation include Iterative Closest Point (ICP) [3], ICP variants [24], and feature-based approaches [26]. But due to the nonuniformity and sparsity of the lidar point clouds, these methods often fail to match such data. ICP approaches find transformations between consecutive scans by minimizing distances between corresponding points from these scans, but points in one frame may miss their spatial counterparts in the next frame, due to sparsity of scan resolution. Feature-based methods are less sensitive to the quality of scans, and hence, are more powerful. However, they are usually more computationally expensive. Furthermore, most feature-based methods are sensitive to another environmental factor, dynamic objects in the scene. These two issues inhibit many feature-based methods from producing effective odometry estimation.

Recently, deep learning-based methods have outperformed classic approaches in many computer vision problems. Many Convolutional Neural Networks (CNNs) architectures and training models have become the state-of-the-art in these tasks. However, the exploration of effective CNNs in some 3D geometric data processing problems, such as 6-DoF pose estimation, has not been this successful yet. Although quite a few CNN-based 6-DoF pose estimation (from RGB images) strategies [34, 43, 36, 39] have been explored recently, these methods often suffer from...
Experiments show that NICP can offer better robustness and count the normal and curvature of each local surface point. Normal Iterative Closest Point (NICP) \[29\] takes into account the spatial relations of scans by applying normal consistency, and better locate effective static regions using a simultaneously learned uncertainty mask.

The main contributions of this paper are as follows: 1) We propose a novel scan-to-scan lidar odometry estimation network which simultaneously learns to estimate the normal and a mask for dynamic regions. 2) We incorporate a spatiotemporal geometry consistency constraint to the network, which provides higher-order interaction between consecutive scans and helps better regularize odometry learning. 3) We couple an efficient mapping module into the estimation pipeline. By utilizing the normal and mask information learned from LO-Net, we can achieve a still real-time but more accurate odometry estimation.

We compared our approach with existing representative methods on commonly used benchmark datasets KITTI \[10, 11\] and Ford Campus Vision and Lidar Data Set \[22\]. Our approach achieves the state-of-the-art results. To our best knowledge, this is the first neural network regression model that is comparable to the state-of-the-art geometry feature based techniques in 3D lidar odometry estimation.

2. Related work

A. ICP variants

Most existing lidar-based pose estimation algorithms are built upon variants of the ICP method \[3, 24\]. ICP iteratively matches the adjacent scans and estimates the pose transformation, by minimizing distances between corresponding elements, until a specific termination condition is met. Despite its wide applicability, ICP is computationally expensive and sensitive to initial poses. Various ICP variants, such as point-to-plane or plane-to-plane ICP, were developed to improve ICP’s convergence speed and robustness against local minima.

By combining point-to-point ICP and point-to-plane ICP, Segal et al. \[28\] introduce a plane-to-plane strategy called Generalized ICP (GICP). The covariance matrices of the local surfaces is used to match corresponding point clouds. Their results show better accuracy over original ICP. The Normal Iterative Closest Point (NICP) \[29\] takes into account the normal and curvature of each local surface point. Experiments show that NICP can offer better robustness and performance than the GICP.

Grant et al. \[12\] derive a novel Hough-based voting scheme for finding planes in laser point clouds. A plane-based registration method \[23\], that aligns detected planes, is adopted to compute the final transformation. For the non-uniformly sampled point clouds from laser sensors, Serafin et al. \[30\] present a fast method to extract robust feature points. They remove ground points by a flat-region-removal method, then extract key-points of lines and planes, and use them to estimate the transformation. They show comparative results against the NARF key-point detector \[31\] and highlight that the feature extraction algorithm is applicable to the Simultaneous Localization and Mapping (SLAM) problem \[13, 5\]. Similarly, Douillard et al. \[8\] remove the ground points by voxelization, cluster the remaining points into the segments, then match these segments through a modified ICP algorithm.

To overcome the sparsity of lidar point clouds, Collar Line Segments (CLS) \[32\] groups the points into polar bins, and generates line segments within each bin. Then, ICP can be used to register the corresponding lines and estimate the transformation between two scans. Although it produces better results than GICP \[28\], CLS is not real-time because the line segments computation is very slow.

Over the past few years, Lidar Odometry And Mapping (LOAM) \[41, 42\] has been considered as the state-of-the-art lidar motion estimation method. It extracts the line and plane features in lidar data, and saves these features to the map for edge-line and plane-surface matching. LOAM dose not consider the dynamic objects in the scene and achieves low-drift and real-time odometry estimation by having two modules running in parallel. The estimated motion of scan-to-scan registration is used to correct the distortion of point clouds and guarantee the real-time performance. Finally, the odometry outputs are optimized through a map.

B. Deep learning-based methods

Deep learning has achieved promising results on the issues of visual odometry (VO) \[34, 43, 40, 36\], image-based pose estimation or localization \[17, 16, 4\], and point cloud classification and segmentation \[25, 19\]. Unfortunately, using deep learning methods to solve 3D lidar odometry problems still remains challenging. Velas et al. \[33\] use CNNs to do lidar odometry estimation on lidar scan sequences. To train the CNNs, the original lidar data is transformed into dense matrix with three channels. However, this model formulates the odometry estimation as a classification problem, rather than a numerical regression, and it only estimates translational parameters. Hence, it is not competent for accurate 6-DoF parameters estimation. The method of \[33\] and ours are conceptually similar, but our network better handles uncertain dynamic regions through mask prediction, and more effectively uses spatiotemporal consistency to regularize the learning for stable estimations.
3. Method

Odometry estimation algorithm uses consecutive point clouds obtained by a moving robot as inputs. Our LO-Net can interpret and exploit more information by performing sequential learning rather than processing a single scan. And the features learned by LO-Net encapsulate geometric information for the LO problem. As shown in Figure 1, our LO-Net architecture consists of a normal estimation sub-network, a mask prediction sub-network, and a Siamese pose regression main-network. LO-Net takes two consecutive scans \((S_{t-1}; S_t)\) as input and jointly estimates the 6-DoF relative pose between the scans, point-wise normal vector, and a mask of moving objects for each scan. Despite being jointly trained, the three modules can be used independently for inference. The odometry outputs of the LO-Net are then refined through a lidar mapping, which registers the lidar point clouds onto a globally constructed map. The final output is the transformation of scan \(S_t\) with respect to the initial reference position, namely, the pose of each \(S_t\) in the world coordinate system.

3.1. Lidar data encoding

As shown in Figure 2, the 3D lidar point clouds with ring structures are usually represented by Cartesian coordinates. Additional information includes intensity values. To convert the original sparse and irregular point clouds into a structured representation that can be fed into networks, we encode the lidar data into point cloud matrices by a cylindrical projection [6]. Given a 3D point \(p = (x, y, z)\) in lidar coordinate system \((X, Y, Z)\), the projection function is

\[
\begin{align*}
\alpha &= \arctan(y/x)/\Delta\alpha \\
\beta &= \arcsin(z/\sqrt{x^2 + y^2 + z^2})/\Delta\beta \\
\end{align*}
\]

where \(\alpha\) and \(\beta\) are the indexes which set the points' positions in the matrix. \(\Delta\alpha\) and \(\Delta\beta\) are the average angular resolution between consecutive beam emitters in the horizontal and vertical directions, respectively. The element at \((\alpha, \beta)\) of the point cloud matrix is filled with intensity value and range value \(r = \sqrt{x^2 + y^2 + z^2}\) of the lidar point \(p\). We keep the point closer to the lidar when multiple points are projected into a same position. After applying this projection on the lidar data, we get a matrix of size \(H \times W \times C\), and \(C\) is the number of matrix channels. An example of the range channel of the matrix is shown in Figure 9.

3.2. Geometric consistency constraint

**Normal estimation.** As shown in Figure 2, given a 3D point \(X^i\) and its \(k\) neighbors \(X^j, j = 1, 2, \ldots, k\) on the grid. The normal \(N(X^i)\) can be estimated by

\[
\arg \min_{N(X^i)} \|w_{i1}(X^i1 - X^i), \ldots, w_{ik}(X^ik - X^i)\|_2^T \cdot N(X^i)\|_2
\]

where \((X^ik - X^i)\) is a 3D vector, \(w_{ik}\) is the weight of \(X^ik\) with respect to \(X^i\), and \(|\cdot|^T\) is a \(k \times 3\) vector. \(\|N(X^i)\|_2 = 1\). We set \(w_{ik} = \exp(-0.2|x(X^ik) - r(X^i)|)\) to put more weight on points which have similar range value \(r\) with \(X^i\), and less weight otherwise. A commonly adopted strategy for solving Equation (2) is to perform a Principal Component Analysis (PCA), and convert the normal estimation to the computation of eigenvectors/eigenvalues of the covariance matrix created from \(X^i\)'s neighbors. In our work, this normal estimation needs to be embedded into the network. The computation of covariance matrices and their eigenvectors makes the training and estimation inefficient. Hence, we simplify the normal estimation by computing the weighted cross products over \(X^i\)'s four neighbors. Then we smooth normal vectors using a moving average filter [20]. This can be formulated as

\[
N(X^i) = \sum_{X^ik, X^ij \in P} (w_{ik}(X^ik - X^i) \times w_{ij}(X^ij - X^i))
\]

where \(P\) is the set of neighboring points of \(X^i\), sorted counterclockwisely, as \(\{X^i1, X^i2, X^i3, X^i4\}\) shown in Figure 2. The final normal vectors are normalized to 1.

Due to the temporal spatial geometric consistency of scan sequences, the points in a point cloud matrix should have the corresponding ones in another. Let \(X^i_{t-1}a\) and \(X^i\beta\) be the spatial corresponding point elements of the consecutive data matrices \(S_{t-1}\) and \(St\), respectively. We can derive \(X^i_{t-1}a\beta\) from \(X^i_{t-1}\) through

\[
\hat{X}^i_{t-1}a\beta = PT_iP^{-1}X^i_{t-1}
\]

where \(T_i\) is the relative rigid pose transformation between the consecutive scans. \(P\) denotes the projection process and \(P^{-1}\) is its inverse operation. Therefore, \(X^i_{t-1}a\beta\) and \(X^i\alpha\beta\) are a pair of matching elements, and we can measure the similarity between the corresponding elements to verify the correctness of pose transformation. The noise inevitably exists in the coordinate and intensity values because of lidar
measurements. We compare the normal $N(x)$ as it reflects smooth surface layouts and clear edge structures of the road environment (see Figure 8). Thus, the constraint of pose transformation can be formulated as minimizing

$$\mathcal{L}_n = \sum_{\alpha \beta} \| N(X_t^{\alpha \beta}) - N(X_t^{\alpha \beta}) \|_1 \cdot e^{\| \nabla r(X_t^{\alpha \beta}) \|}$$

(5)

where $\nabla r(X_t^{\alpha \beta})$ is a local range smooth measurement, and $\nabla$ is the differential operator with $\alpha$ and $\beta$. The item $e^{\| \cdot \|}$ allows the loss function to focus more on sharply changing areas in the scene.

### 3.3. Lidar odometry regression

To infer the 6-DoF relative pose between the scans $S_{t-1}$ and $S_t$, we construct a two-stream network. The input to the network is encoded point cloud matrices with point-wise normal vectors. As shown in Figure 1, LO-Net concatenates the features coming from two individual streams of feature extraction networks, then pass these concatenated features to the following four convolution units. The last three fully-connected layers output the 6-DoF pose between the input scans. The last two fully-connected layers are of dimensions 3 and 4, for regressing the translation $x$ and rotation quaternion $q$, respectively. Finally, we get a 7D vector, which can be converted to a 4 x 4 transformation matrix.

In order to reduce the number of model parameters and computation cost, we replace most of convolution layers of odometry regression network with fireConv [15], which has been used for object detection from lidar point clouds [35]. We follow [15] to construct our feature extraction layers. Since the width of intermediate features is much larger than its height, we only down-sample the width by using max-pooling during feature extraction. The detailed network architecture is illustrated in the supplementary material.

**Learning position and orientation simultaneously.** In our method, we choose the quaternion to represent the rotation as it is a continuous and smooth representation of rotation. The quaternions $q$ and $-q$ map to the same rotation, so we need to constrain them to a unit hemisphere. We use $L_x(S_{t-1}; S_t)$ and $L_q(S_{t-1}; S_t)$ to demonstrate how to learn the relative translational and rotational components, respectively.

$$L_x(S_{t-1}; S_t) = \| x_t - \hat{x}_t \|_1 $$

$$L_q(S_{t-1}; S_t) = \| q_t - \hat{q}_t \|_1$$

(6)

where $x_t$ and $q_t$ are the ground truth relative translational and rotational components, $\hat{x}_t$ and $\hat{q}_t$ denote their predicted counterparts. $l$ refers to the distance norm in Euclidean space, and we use the $\ell_2$-norm in this work. Due to the difference in scale and units between the translational and rotational pose components, previous works [17, 34] gave a weight regularizer $\lambda$ to the rotational loss to jointly learn the 6-DoF pose. However, the hyper-parameter $\lambda$ need to be tuned when using new data from different scene. To avoid this problem, we use two learnable parameters $s_x$ and $s_q$ to balance the scale between the translational and rotational components in the loss term [16].

$$L_o = L_x(S_{t-1}; S_t) \exp(-s_x) + s_x$$

$$+ L_q(S_{t-1}; S_t) \exp(-s_q) + s_q$$

(7)

We use the initial values of $s_x = 0.0$ and $s_q = -2.5$ for all scenes during the training.

### 3.4. Mask prediction

Given two consecutive scans $S_{t-1}$ and $S_t$ in a static rigid scene, we can get point matching relationships of the encoded data matrix pairs through transformation and cylindrical projection, as illustrated in Section 3.2. Lidar point clouds are considered as the 3D model of the scene, and often contain dynamic objects such as cars and pedestrians in the road environment. These factors may inhibit the learning pipeline of odometry regression.

Based on the encoder-decoder architecture (see Figure 1), we deploy a mask prediction network [43, 38] to learn the compensation for dynamic objects, and improve the effectiveness of the learned features and the robustness of the network. The encoding layers share parameters with feature extraction layers of the odometry regression network, and we train these two networks jointly and simultaneously. The deconvolution layers are variants of fireDeconv [15], and adopt the skip-connection. The detailed network architecture is described in the supplementary material.

The predicted mask $M(X_t^{\alpha \beta}) \in [0, 1]$ indicates the area where geometric consistency can be modeled or not, and implicitly ensures the reliability of the features learned in the pose regression network. Therefore, the geometric consistency error as formulated in Equation (5) is weighted by

$$L_n = \sum_{\alpha \beta} M(X_t^{\alpha \beta}) \| N(X_t^{\alpha \beta}) - N(X_t^{\alpha \beta}) \|_1 \cdot e^{\| \nabla r(X_t^{\alpha \beta}) \|}.$$  

(8)

Note that there is no ground truth label or supervision to train the mask prediction. The network can minimize the loss by setting all values of the predicted mask to be 0. To avoid this, we add a cross-entropy loss as a regularization term

$$L_r = - \sum_{\alpha \beta} \log P(M(X_t^{\alpha \beta}) = 1).$$

(9)

In summary, our final objective function to minimize for odometry regression is

$$L = L_o + \lambda_o L_n + \lambda_r L_r$$

(10)
where $\lambda_n$ and $\lambda_r$ are the weighting factors for geometric consistency loss and mask regularization, respectively.

### 3.5. Mapping: scan-to-map refinement

Consecutive scan-to-scan matches could introduce accumulative error, and also, may suffer when available common feature points between consecutive frames are limited. Hence, we maintain a global map reconstructed from the previously scans, then use the registration between this map and the current scan to refine the odometry. Unlike the traditional scan-to-map approaches [41, 42], which directly match all the detected edge and plane points, we use the normal information (estimated by LO-Net) to select points from smooth area and use the mask (also an output of LO-Net) to exclude points from moving object.

At a test time $t$, let $[T_i, S_t]$ be the data from LO-Net. $T_i$ is the odometry computed by LO-Net, which is used as the initial pose for mapping. $S_t$ is a multi-channel data matrix containing the intensity, range, normal, and mask values of each point. The coordinates of each point can be calculated from its range value. The mapping takes the scan and the current scan to refine the odometry. Unlike the traditional scan-to-map approaches, which directly match all the detected edge and plane points, we use the normal information (estimated by LO-Net) to select points from smooth area and use the mask (also an output of LO-Net) to exclude points from moving object.

The mapping takes the scan and the odometry as input, then matches the point cloud onto the global map. Figure 3 shows the diagram of our mapping module. The details of diagram are as follows:

- **$\hat{\Theta}$**: Iterate register the scan onto the map by solving Equation (12) until a maximum number of iteration $n_{iter}$. Then, calculate the final transformation $M_t$ by accumulating the transformation during the iteration $\hat{M}_k$ and the initial pose $M_{init}$

$$M_t = \prod_{k=1}^{n_{iter}} \hat{M}_k M_{init}. \tag{13}$$

- **$\Phi$**: Generate a new point cloud from the current scan $S_t$ by linear interpolation of vehicle motion between $M_{t-1}$ and $M_t$.

- **$\Sigma$, N**: Add the new point cloud to the map. Then, remove the oldest point cloud scans to only maintain $n_m$ transformed scans in the map.

This mapping-based refinement is performed iteratively along with the scan sequence. It can further improve the accuracy of odometry estimation, as shown in Section 4.2.

### 4. Experiments

#### Implementation details.

The point cloud data we use is captured by the Velodyne HDL-64 3D lidar sensor (64 laser beams, 10Hz, collecting about 1.3 million points/second). Therefore, during encoding the data matrix, we set $H = 64$ and $W = 1800$ by considering the sparseness of point clouds. The width of input data matrix are resized to 1792 by cropping both ends of the matrix. During the training, the length of input sequence is set to be 3. We form the temporal pairs by choosing scan $[S_{t-2}, S_{t-1}, S_{t-1}, S_t]$, $[S_{t-1}, S_t]$, $[S_{t-2}, S_t]$. LO-Net predicts relative transformation between the pairs. The whole framework is implemented with the popular Tensorflow library [1]. During the training, the mask prediction network is pre-trained using KITTI 3D object detection dataset, and all layers are trained simultaneously. The initial learning rate is 0.001 and exponentially decays after every 10 epochs until 0.00001. The loss weights of Equation (10) are set to be $\lambda_n = 0.15$ and $\lambda_r = 0.05$, and the batch size is 8. We
choose the Adam [18] solver with default parameters for optimization. The network is trained on an NVIDIA 1080 Ti GPU. For the mapping, we set the number of points in each scan $n_x = 0.01HW$, the number of scans in the map $n_m = 100$, and the number of iterations $n_{iter} = 15$.

4.1. Datasets

**KITTI.** The KITTI odometry dataset [10, 11] consists of 22 independent sequences with stereo gray-scale and color camera images, point clouds captured by a Velodyne lidar sensor, and calibration files. Sequences 00-10 (23201 scans) are provided with ground truth pose obtained from the IMU/GPS readings. For sequences 11-21 (20351 scans), there is no ground truth available, and are provided for benchmarking purposes. The dataset is captured during driving in a variety of road environments with dynamic objects and vegetation, such as highways, country roads and urban areas. The driving speed is up to 90km/h.

**FORD.** The Ford Campus Vision and Lidar Data Set [22] consists time-synchronized 2D panoramic image, 3D lidar point clouds and IMU/GPS data. Like KITTI, the lidar dataset is captured using horizontally scanning 3D lidar mounted on the top of a vehicle. The dataset contains two loop closure sequences collected in different urban environments, and there are more moving vehicles than the KITTI dataset.

4.2. Odometry results

**Baselines.** We compare our approach with several classic lidar odometry estimation methods: ICP-point2point (ICP-po2po), ICP-point2plane (ICP-po2pl), GICP [28], CLS [32], LOAM [42] and Velas et al. [33]. The first two ICP methods are implemented using the Point Cloud Library [27]. As far as we know, [33] is the only deep learning based lidar odometry method that has comparable results, but it has no codes publicly available. We obtain the results of other evaluated methods using the publicly available code, and some of the results are even better than those in the originally published papers. For LOAM algorithm, it achieves the best results among lidar-based methods in the KITTI odometry evaluation benchmark [11]. In order to enable the map-based optimization of LOAM to run for every input scan and determine the full potential of the algorithm, we make some modifications and parameter adjustments in the originally published code. Loop closure detection is not implemented for all methods since we aim to test the limits of accurate odometry estimation.

We firstly conduct the training and testing experiments on the KITTI dataset. Then, based on the model trained only on the KITTI dataset, we directly test the model on the Ford dataset without any further training or fine-tuning. We use the KITTI odometry evaluation metrics [11] to quantitatively analyze the accuracy of odometry estimation. Table 1 shows the evaluation results of the methods on KITTI and Ford datasets. It can be seen that the results of LO-Net+Mapping are slightly better than LOAM and clearly superior to the others. Although there are differences between the two datasets, such as different lidar calibration parameters and different systems for obtaining ground truth, our approach still achieves the best average performance among evaluated methods. Figure 5 shows the 3D trajectory plots of our method at different stages. Some trajectories produced by different methods are shown in Figure 4 and 6. Figure 7 shows the average evaluation errors on KITTI Seq. 00-10. More estimated trajectories for KITTI and Ford datasets are shown in our supplementary material.

**Ablation study.** In order to investigate the importance of different loss components proposed in Section 3, we perform an ablation study on KITTI dataset by training and testing the LO-Net using different combinations of the losses. As shown in Table 2, the network achieves the best average performance when it is trained with the full loss.

4.3. Normal results

Since KITTI and Ford datasets do not provide a benchmark for normal evaluation, we compare the normal esti-
neighboring points. In our experiment, we choose the radius point by fitting a least-square plane from its surrounding information with that computed from the methods of PCA and Holzer [14]. The PCA estimates the surface normal at a point by fitting a least-square plane from its surrounding neighboring points. In our experiment, we choose the radius

Table 1. Odometry results on KITTI and Ford datasets. Our network is trained on KITTI sequences and then tested on the two datasets.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>r&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>t&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>r&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>t&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>r&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>t&lt;sub&gt;rel&lt;/sub&gt;</td>
<td>r&lt;sub&gt;rel&lt;/sub&gt;</td>
</tr>
<tr>
<td>00&lt;sup&gt;†&lt;/sup&gt;</td>
<td>6.88</td>
<td>2.99</td>
<td>3.80</td>
<td>1.73</td>
<td>1.29</td>
<td>0.64</td>
<td>2.11</td>
<td>0.95</td>
</tr>
<tr>
<td>01&lt;sup&gt;†&lt;/sup&gt;</td>
<td>11.21</td>
<td>2.58</td>
<td>13.53</td>
<td>2.58</td>
<td>4.39</td>
<td>0.91</td>
<td>4.22</td>
<td>1.05</td>
</tr>
<tr>
<td>02&lt;sup&gt;†&lt;/sup&gt;</td>
<td>8.21</td>
<td>3.39</td>
<td>9.00</td>
<td>2.74</td>
<td>2.53</td>
<td>0.77</td>
<td>2.29</td>
<td>0.86</td>
</tr>
<tr>
<td>03&lt;sup&gt;†&lt;/sup&gt;</td>
<td>11.07</td>
<td>5.05</td>
<td>2.72</td>
<td>1.63</td>
<td>1.68</td>
<td>1.08</td>
<td>1.63</td>
<td>1.09</td>
</tr>
<tr>
<td>04&lt;sup&gt;†&lt;/sup&gt;</td>
<td>6.64</td>
<td>4.02</td>
<td>2.96</td>
<td>2.58</td>
<td>3.76</td>
<td>1.07</td>
<td>1.59</td>
<td>0.71</td>
</tr>
<tr>
<td>05&lt;sup&gt;†&lt;/sup&gt;</td>
<td>3.97</td>
<td>1.93</td>
<td>2.29</td>
<td>1.08</td>
<td>1.02</td>
<td>0.54</td>
<td>1.98</td>
<td>0.92</td>
</tr>
<tr>
<td>06&lt;sup&gt;†&lt;/sup&gt;</td>
<td>1.95</td>
<td>1.59</td>
<td>1.77</td>
<td>1.00</td>
<td>0.92</td>
<td>0.46</td>
<td>0.92</td>
<td>0.46</td>
</tr>
<tr>
<td>07&lt;sup&gt;†&lt;/sup&gt;</td>
<td>5.17</td>
<td>3.35</td>
<td>1.55</td>
<td>1.42</td>
<td>0.64</td>
<td>0.45</td>
<td>1.04</td>
<td>0.73</td>
</tr>
<tr>
<td>08&lt;sup&gt;†&lt;/sup&gt;</td>
<td>10.04</td>
<td>4.93</td>
<td>4.42</td>
<td>2.14</td>
<td>1.58</td>
<td>0.75</td>
<td>2.14</td>
<td>1.05</td>
</tr>
<tr>
<td>09&lt;sup&gt;†&lt;/sup&gt;</td>
<td>6.93</td>
<td>2.89</td>
<td>3.95</td>
<td>1.71</td>
<td>1.97</td>
<td>0.77</td>
<td>1.95</td>
<td>0.92</td>
</tr>
<tr>
<td>10&lt;sup&gt;†&lt;/sup&gt;</td>
<td>8.91</td>
<td>4.74</td>
<td>6.13</td>
<td>2.60</td>
<td>1.31</td>
<td>0.62</td>
<td>3.46</td>
<td>1.28</td>
</tr>
</tbody>
</table>

In Table 1, we provide the mean values of translational and rotational RMSE on KITTI training and testing sequences (mean<sup>†</sup>, mean<sup>‡</sup>), and the results on Ford dataset are taken from [33]. The results on KITTI dataset outside the brackets are obtained by running the code, and those in the brackets are taken from [42]. The sequences of KITTI dataset that are not used to train LO-Net<sup>‡</sup>, the average errors of translation and rotation with respect to path length intervals. Our LO-Net+Mapping produces most accurate trajectory.

Table 2. Comparison of different combinations of the losses. The mean values of translational and rotational RMSE on KITTI training and testing sequences are computed as in Table 1. $\epsilon_{n}$ indicates that the geometric consistency loss is not weighted by the mask.

<table>
<thead>
<tr>
<th>Seq.</th>
<th>$L_{o}$</th>
<th>$L_{o}$&lt;sup&gt;′&lt;/sup&gt;</th>
<th>$L_{o}$, $L_{n}$, $L_{r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_{rel}$</td>
<td>$r_{rel}$</td>
<td>$t_{rel}$</td>
</tr>
<tr>
<td>mean&lt;sup&gt;†&lt;/sup&gt;</td>
<td>1.46</td>
<td>1.01</td>
<td>1.18</td>
</tr>
<tr>
<td>mean&lt;sup&gt;‡&lt;/sup&gt;</td>
<td>2.03</td>
<td>1.50</td>
<td>1.80</td>
</tr>
</tbody>
</table>

$r = 0.5$m and $r = 1.0$m as the scale factor to determine the set of nearest neighbors of a point. As shown in Figure 8, our estimated normals can extract smooth scene layouts and clear edge structures.

For quantitative comparison purpose, the normals computed from PCA with $r = 0.5$m are interpolated and used as the ground truth. Then a point-wise cosine distance is employed to compute the error between the predicted normal and the ground truth.

$$e_{i} = \arccos(\hat{n}_{i} \cdot \hat{n}_{i})$$

where the angle $e_{i}$ is the normal error at point $p_{i}$, $\hat{n}_{i}$ and $\hat{n}_{i}$ are ground truth and predicted normal vector of point $p_{i}$, respectively. $\hat{n}_{i} \in N$ denotes the point $p_{i}$ is a valid point with ground truth normal. The normal evaluations performed on KITTI dataset are shown in Table 3, our approach outperforms the others under most metrics. The metrics include the mean and median values of $e_{i}$, and the percent-good-normals whose angle fall within the given thresholds [9, 37]. “GT median” which denotes that we set a normal direction for all points with the median value of the ground truth, is employed as a baseline. The evaluation demonstrates that
4.4. Mask visualization

Examples of the mask predicted by our network are visualized in Figure 9. The highlighted areas suggest that LO-Net has learned to identify dynamic objects and tend to mask vegetation as unexplainable, and indicate that the network will pay less attention to these areas in the odometry regression. Dynamic objects and the relationships between scan sequences are important for odometry estimation problems. They are difficult to explicitly model but implicitly learned by our network.

4.5. Runtime

The lidar scanning point cloud are captured one by one over time, and processing these data in time is critical for robotic applications. Note that unlike image-based computer vision applications, the commonly used lidar sensors, such as Velodyne HDL-64 used in the KITTI and Ford dataset, rotate at a rate of 10Hz, that is 0.1s per-scan. Therefore, the real-time performance here means that the processing time of each scanning data is less than 0.1s. An NVIDIA 1080 Ti GPU and an Intel Core i7 3.4GHz 4-core CPU are chose as our test platform. In test-time, the data batch-size of LO-Net is set to 1. Table 4 shows the average running times on Seq. 00 of KITTI dataset. The average processing time of our framework is about 80.1ms per-scan in total. Reasonably, most of our runtime is spent on the mapping procedure. Compared with most of traditional lidar odometry estimation methods, including the methods evaluated in Section 4.2, our map-based optimization is lightning fast since we consume a new representation of input data. Our approach enables real-time performance through a straightforward pipeline on a platform with GPU. For lower performance platforms, we can also speed up the processing through the parallelism of LO-Net and the mapping. Currently, some parts of our framework run on CPU, and we can implement them on GPU to further increase the speed.

5. Conclusions

We present a novel learning framework LO-Net to perform lidar odometry estimation. An efficient mapping module is coupled into the estimation framework to further improve the performance. Experiments on public benchmarks demonstrate the effectiveness of our framework over existing approaches.

There are still some challenges that need to be addressed: 1) The point clouds are encoded into data matrices to feed into the network. Direct processing of 3D point clouds could be more practical for 3D visual tasks. 2) Our current network is trained with ground truth data. This limits the application scenarios of the network. In our future work, we will investigate in more detail the geometry feature representation learned by the network. We also plan to incorporate recurrent units into this network to build temporal-related features. This may lead to an end-to-end framework without the need of costly collections of ground truth data.

Acknowledgment

This work is supported by National Natural Science Foundation of China (No. U1605254, 61728206), and the National Science Foundation of USA under Grants EAR-1760582.
References


