SelfVoxeLO: Self-supervised LiDAR Odometry with Voxel-based Deep Neural Networks

Yan Xu[1] **Zhaoyang Huang**[1,2] **Kwan-Yee Lin***[1,3] **Xinge Zhu**[1] **Jianping Shi**[3]
Hujun Bao[2] **Guofeng Zhang**[2] **Hongsheng Li**[1]
[1]Multimedia Laboratory, The Chinese University of Hong Kong
[2]State Key Lab of CAD&CG, Zhejiang University [3]SenseTime Research

Abstract: Recent learning-based LiDAR odometry methods have demonstrated their competitiveness. However, most methods still face two substantial challenges: 1) the 2D projection representation of LiDAR data cannot effectively encode 3D structures from the point clouds; 2) the needs for a large amount of labeled data for training limit the application scope of these methods. In this paper, we propose a self-supervised LiDAR odometry method, dubbed SelfVoxeLO, to tackle these two difficulties. Specifically, we propose a 3D convolution network to process the raw LiDAR data directly, which extracts features that better encode the 3D geometric patterns. To suit our network to self-supervised learning, we design several novel loss functions that utilize the inherent properties of LiDAR point clouds. Moreover, an uncertainty-aware mechanism is incorporated in the loss functions to alleviate the interference of moving objects/noises. We evaluate our method's performances on two large-scale datasets, *i.e.*, KITTI and Apollo-SouthBay. Our method outperforms state-of-the-art unsupervised methods by 27%/32% in terms of translational/rotational errors on the KITTI dataset and also performs well on the Apollo-SouthBay dataset. By including more unlabelled training data, our method can further improve performance comparable to the supervised methods.

Keywords: Odometry, 3D vision, Deep learning

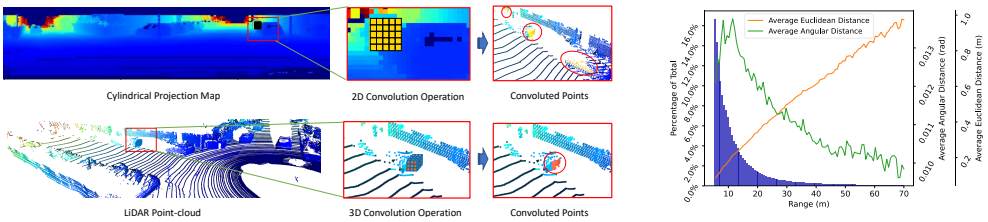
1 Introduction

Ego-motion estimation from temporal sequences of sensor data, also known as odometry, is of fundamental importance for many robotic vision tasks, including navigation, mapping, virtual/augmented reality, *etc*. Compared with visual sensors, the LiDAR can capture richer 3D geometric information of the environments and is robust against varying lighting conditions. Hence, a reliable LiDAR odometry system is desirable for localization systems.

The classic methods [1, 2, 3, 4, 5] are mainly based on the point registration and work well in ideal scenarios, but they might fail in practice due to the sparse nature of point clouds and environmental noises. Typically, ICP [1] and its variants [1, 2, 3] iteratively find the point correspondences with nearest-neighbor searching and optimize for the pose transformations. This optimization procedure ignoring the correspondence reliability can easily run into local optimum especially when noise and dynamic objects exist. In the past a few years, the advances in deep learning have significantly advanced state of the arts in odometry estimation. Seminal works [6, 7, 8], demonstrate the feasibility of 6-DOF pose regression via convolutional neural network for visual odometry estimation. Following the visual pipelines, several LiDAR odometry estimation approaches have been proposed [9, 10, 11], where they often project the 3D LiDAR points onto a cylindrical surface and then adopt the similar framework from CNN-based visual odometry methods. However, projecting the point clouds to the cylindrical surface inevitably alters the 3D topology and cannot effectively capture the geometric information. As illustrated in Fig. 1a, the 2D convolution on the cylindrical

* Kwan-Yee Lin is the corresponding author.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.



(a) (b)

Figure 1: (a) The difference between the 2D convolution and the 3D convolution. (b) The distribution of point clouds with respect to LiDAR range value, and the measured average distances with different metrics (*i.e.*, Euclidean distance and angular distance [14]) in each bin. Large Euclidean distances in distant regions make the unreliable remote points count more, which is not desirable.

projection map might process the points on objects far away from each other, which ignore the 3D topology relations and contaminate the encoded features. A straightforward idea to tackle the issue of cylindrical projection is to adopt 3D representations and process the points in the 3D space directly. Therefore, the 3D convolution network is an appealing alternative. In contrast to the 2D convolution, the 3D convolution can better retain the 3D topology relations and structures during the hierarchical feature extraction. On the other hand, it should be noted that the significant progress of CNN-based methods relies, to a large extent, on large-scale annotated training data, which is not always feasible in practice, due to the huge cost needed for large-scale annotations. How to train a LiDAR odometry network in an unsupervised manner is still an imperative problem.

In this work, we develop an self-supervised learning-based LiDAR odometry method with 3D convolution networks. Specifically, our network first voxelizes the point clouds into fine-grid voxel cells, and extracts 3D features via 3D convolutional neural networks. Then, the extracted features are fed into our odometry regression network to predict the final 6-DOF ego-motions. To suit our network to self-supervised training, we analyse the inherent properties of the LiDAR point clouds and propose several losses: the spherical reprojection loss that essentially pushes the network to focus on the nearby stable points, the transformation residual loss to stabilize the training, and the deep flow supervision loss to facilitate the point-wise feature learning. Furthermore, to mitigate the interference from noise and dynamic objects, we also propose to estimate the correspondence-pair confidences, which are incorporated into our self-supervised losses to set lower weights for the unreliable point correspondences.

Our contributions can be summarized as follows: (1) We abandon the common 2D projection-based LiDAR odometry framework and investigate the effectiveness of voxel-based 3D geometric representation. We propose a framework that predicts ego-motions from raw LiDAR points based on 3D convolutional neural networks, which are trained in an unsupervised manner. (2) Several self-supervised loss functions are introduced, *i.e.*, the spherical reprojection loss, the range alignment loss, the transformation residual loss and the deep flow supervision loss to train ego-motion prediciton without any annotations. Furthermore, we incorporate an uncertainty-aware mechanism into the loss functions to mitigate the interference of moving objects and noise. (3) The proposed method achieves state-of-the-art performances on two public odometry datasets, *i.e.*, the KITTI dataset [12] and the Apollo-SouthBay Dataset [13].

2 Related Work

Odometry. LiDAR odometry algorithms can be categorized as two-frame methods [1, 2, 3] and multi-frame methods [4, 5]. The classic two-frame methods are mostly based on the point registration, where ICP [1] and its variants [15, 2, 3, 16] are typical exemplars. The ICP iteratively finds the point correspondences and optimizes for the pose transformation between two LiDAR pointclouds util convergence. Different ICP methods may apply different weights on the correspondence pairs during optimization according to the geometric characteristics. However they often fail to model the moving objects that violate the algorithm assumption. Moreover, most of these methods are too computationally expensive to be applied in real-time systems. The multi-frame algorithms (often referred as mapping [4, 5, 17]) are often used to refine the two-frame based estimation by incorporating more frames into optimization. They are computationally heavier and usually runs in the backend at a lower frequency. In our work, we mainly focus on the two-frame method which is more fundamental and can be combined with multi-view methods. Recently, many CNN-based odometry methods

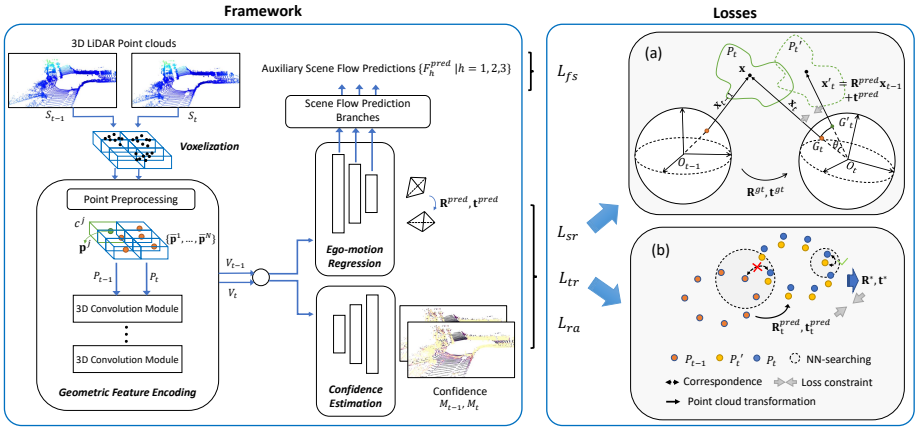


Figure 2: Framework overview and loss functions. The input LiDAR point clouds are voxelized and then fed to the 3D ***geometric feature encoding*** module, where the points are pre-downsampled (obtaining point sets P_t and P_{t-1}) and encoded by the 3D convolution modules to obtain the 3D geometric feature sets V_{t-1} and V_t. Based on these 3D features, the ***ego-motion regression*** module and the ***confidence estimation*** module predict the ego-motion $(\mathbf{R}^{pred}, \mathbf{t}^{pred})$ and the voxel-wise confidence estimations (M_{t-1} and M_t) for each sweep. The sub-figure (a) and (b) demonstrate our spherical reprojection loss and transformation residual loss, which are elaborated in Sec. 3.2.

were proposed. Initially, several seminal methods were proposed for visual odometry [6, 7, 8]. More recently, researchers used CNN in LiDAR odometry [10, 9]. They represented the LiDAR pointclouds by cylindrical projection and then borrowed the network architectures from the visual odometry methods. Cho *et al*. [11] extended this pipeline to unsupervised learning inspired by the unsupervised visual odometry [8, 18]. However, the cylindrical projection may lose the spatially geometric structures of the input pointclouds, which leads to unreliable feature extraction.

3D network. In the past few years, various 3D DNNs have been proposed to better handle the 3D data. The 3D convolution operation is also applied on local areas centered at fixed grids, but the operation are conducted in the 3D space rather than the 2D space. PointNet [19, 20] determines the grid locations with furthest sampling and find the related feature points in respective local areas with nearest-neighbor searching. It achieves high flexibility but sacrifices speed. To achieve high efficiency, Zhou *et al*. [21] proposed to split the space into voxels uniformly and apply the 3D convolution on these uniform 3D grids. Gu *et al*. [22] proposed the 3D convolution on permutohedral lattices. Hanocka *et al*. [23] proposed a new set of convolution operations to handle 3D mesh data. Compared with 2D convolution, 3D convolution is more suitable to process the 3D data and has made great success in 3D object detection [21, 24, 25, 26, 27], 3D scene flow predicition[22, 28] and 3D semantic segmentaion [19, 20, 23, 29]. However, few investigation has been made using 3D convolutional networks on LiDAR odometry.

3 Method

To tackle the mentioned challenges, *i.e*., the limitation of 2D convolution and the need of large-scale labeled data, our framework aims to estimate the 6-DOF agent's pose transformation $(\mathbf{R}^{pred}, \mathbf{t}^{pred})$ from two successively scanned point clouds S_{t-1} and S_t, *without ground-truth transformations*. Different from the previous works [10, 9, 11] adopting 2D convolutions on cylindrical projection maps of the point clouds, we use fine-grid voxels to represent the LiDAR point cloud and propose to directly process the point clouds with 3D convolutions, which can better maintain the 3D geometric information than the 2D convolutions in existing projection-based methods. To enable stable training of LiDAR-based odometry without ground-truth, we introduce three self-supervised losses and uncertainty-aware mechanism. As shown in Fig. 5, our framework mainly consists of three modules: (1) 3D geometric feature encoding, (2) ego-motion regression and (3) confidence estimation. We will detail each part and the proposed unsupervised loss functions in the ensuing sections.

3.1 3D Geometric Feature Encoding

The odometry task requires real-time response as well as high precision. As mentioned above, the 3D convolution on 3D feature spaces can efficiently preserve the geometric information from

the scene, which is an appealing candidate. However, the 3D space is unbounded with low data occupancy. Directly applying 3D convolutions to 3D point clouds is resource intensive. Previous works make a compromise and project the LiDAR point clouds to cylindrical maps to persist with the legacy of 2D CNNs, which obviously is not an optimal solution. Recently, with the development of 3D CNNs, many efficient 3D operators spring up making efficient 3D geometric feature encoding possible. Hence, to achieve precise feature encoding while maintaining real-time speed, we conduct 3D feature encoding in our framework by adopting the voxel representation and the recent proposed sparse submanifold convolutions [30].

Voxelization. Let $\mathcal{C}$ denotes the 3D overall space with sizes of (D, W, H) along the axes of x, y and z. As illustrated in Fig.5, we first divide the space into equal-sized cells c^i with sizes of $(D/n_D, W/n_W, H/n_H)$. Then, the points $\mathbf{p}_i$'s in the point cloud P are dispensed to the respective cells and we refer to these cells as voxels in the following. Due to the sparse nature of LiDAR data, most voxels are only allocated 2-3 points with the voxel size of (10 cm, 10 cm, 20 cm), which achieves a good balance between representation precision and resource cost.

Feature Encoding. Each LiDAR point $\mathbf{p}^i \in S$ is a vector of $[x^i, y^i, z^i, n_x^i, n_y^i, n_z^i, r^i]^T$, where (x^i, y^i, z^i) is the i-th point's coordinate, (n_x^i, n_y^i, n_z^i) is its normal vector (which can be obtained via cross products over four neighbors similar to [10, 11]) and r_i stands for the reflectance value. For computational efficiency, we compute the arithmetic average $\bar{\mathbf{p}}^j = 1/|c^j| \sum_{\mathbf{p}^i \in c^j} \mathbf{p}^i$ for each voxel as its representation and obtain down-sampled point clouds P_t and P_{t-1}, which will be used as the initial point cloud's representation. Thereafter, we extract the high level features from the point cloud through stacked 3D submanifold convolutions [30] illustrated in Fig. 5. The 3D submanifold convolution, combined of valid sparse convolution and sparse convolution, has the advantages of accurately capturing 3D local geometric patterns of the input point cloud while ignoring the empty voxels to accelerate the encoding process. The 3D convolution maintains the geometric structures and spatial topology during down-sampling/up-sampling, which is quite challenging for the 2D convolution as shown in Fig. 1a. Finally, we obtain the voxel feature volumes V_{t-1} and V_t as an intermediate high-level representation for the point clouds P_{t-1} and P_t respectively.

3.2 Odometry Regression

After encoding the voxel features V_{t-1} and V_t from the input point clouds, we design a network to predict the ego-motion from these features. To reduce the memory cost and improve the computational efficiency, we follow [21] to reshape the 3D features volume of the two timestamps $t-1$ and t to the 2D bird-view feature maps respectively and channel-wisely concatenate the features of the two timestamps before feeding them into the ego-motion regression module. Thanks to the powerful encoding capability of 3D CNNs, the features can successfully encode the agent's motion in both the ground plane and the vertical direction.

The ego-motion regression module is constituted by several ResNet [31] blocks followed by fully connected layers, which estimates the ego-motions $(\mathbf{R}^{pred}, \mathbf{t}^{pred})$ as illustrated in Fig. 5. In our implementation, we let the network predict the quaternion vector, a more compact representation for rotation, which is more suitable as the regression target. Since the quaternion representation and the matrix representation are equivalent, we uniformly refer to the rotation prediction as $\mathbf{R}^{pred}$ in the following for simplicity.

Spherical Reprojection Loss. Let O_{t-1} and O_t be the agent coordinate systems at two consecutive timestamps and $\mathbf{x}$ denote an arbitrary static point in the 3D scene. As illustrated in Fig 5a. The observations of this point in systems O_{t-1} and O_t are denoted as $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ respectively. Ideally, we can obtain $\mathbf{x}_t$ as

$$\mathbf{x}_t = \mathbf{R}^{gt}\mathbf{x}_{t-1} + \mathbf{t}^{gt}, \tag{1}$$

where $\mathbf{R}^{gt}$ and $\mathbf{t}^{gt}$ are the ground-truth ego-motion, which however is not available in our setup. To achieve self-supervised learning of ego-motion prediction, our network leverages the geometric consistency among frames: to first transform the point cloud P_{t-1} with the current prediction $(\mathbf{R}^{pred}, \mathbf{t}^{pred})$ obtaining P'_t, *i.e.*, transform each point $\mathbf{x}_{t-1} \in P_{t-1}$ as $\mathbf{x}'_t = \mathbf{R}^{pred}\mathbf{x}_{t-1} + \mathbf{t}^{pred}$, and then minimize the distance between P'_t and P_t to push the prediction closer to $(\mathbf{R}^{gt}, \mathbf{t}^{gt})$. The nearest-neighbor Euclidean distance is a common choice to point-wisely measure the deviation between the two point clouds. However, we find that the Euclidean distance is not an optimal choice to directly measure the discrepancy between the two 3D point clouds, because of the high sparsity level and less measurement accuracy in the distance caused by the sparsity nature of the LiDAR

point clouds. Due to the noise and the erroneous correspondence identifying caused by the increasing sparsity level, the nearest-neighbor Euclidean distances between the two point clouds are much larger in distant regions than those of the nearby regions, which are illustrated in Fig. 1b. As a result, the distant unreliable points might contribute more in the loss function with Euclidean distance measurement, which is not desirable. Inspired by the reprojection error [32] widely used in visual odometry, we propose the spherical reprojection loss to make the loss focusing more on the nearby reliable points. Specifically, we adopt the pair-wise angular distance to measure the distance between the associated points with nearest-neighbor searching from the point clouds P_t and P'_t, where the angular distance is defined as the angle between origin-to-point rays of associated correspondence points in P_t and P_{t-1}. As shown in Fig. 5a, the angular distance between the nearest-neighboring points $\mathbf{x}_t$ and $\mathbf{x}'_t$ is defined as the angle θ between the rays O_tG_t and $O_tG'_t$, where G_t and G'_t are the projection of nearest-neighboring points on a unit sphere centered at O_t. The minimization of this angle error is equivalent to minimize the geodesic distance on the sphere, which is similar to the minimization of reprojection error. For numeric stability, we minimize the $-\cos(\theta)$, a monotonically increasing function of θ in $[0, \pi]$. The spherical reprojection loss is hence expressed as

$$L_{sr} = -\frac{1}{|P_t|} \sum_{\mathbf{x}_t^i \in P_t, j=\mathcal{M}(i)} \left(\frac{\mathbf{x}_t^i \cdot \mathbf{x}_t'^j}{||\mathbf{x}_t^i||_2 ||\mathbf{x}_t'^j||_2} \right), \tag{2}$$

where $\mathcal{M}(\cdot)$ denotes the nearest-neighbor identifying process between the points in P_t and P'_t. As illustrated by the green curve in Fig. 1b, the nearby errors contribute more to the spherical reprojection loss, which makes the loss focus more on the nearby reliable regions.

Transformation Residual Loss. To stabilize and speed up the convergence, we further incorporate the classic ICP algorithm [33, 34] into the loss function to directly guide the ego-motion estimation learning. As illustrate in Fig. 5b, we first align the point cloud P_{t-1} to P'_t with the current ego-motion prediction as in the previous section, and then calculate transformation residual $(\delta\mathbf{R}, \delta\mathbf{t})$ from P'_t to P_t with ICP iteration. The ICP iteratively finds the nearest-neighbor correspondences $(\mathbf{x}_t^i, \mathbf{x}_t'^j)$'s and minimize the point-to-plane distances $\mathcal{D}(\mathbf{x}_t^i, \mathbf{x}_t'^j)$ between them. By accumulating the computed transformation residual to the current prediction ($\mathbf{R}^{pred}$,$\mathbf{t}^{pred}$), we can obtain a more accurate ego-motion:

$$\mathbf{t}^* = \delta\mathbf{R}\mathbf{t}^{pred} + \delta\mathbf{t}, \quad \mathbf{R}^* = \delta\mathbf{R}\mathbf{R}^{pred} \tag{3}$$

To improve the prediction accuracy, we design our transformation residual loss as

$$L_{tr} = u_\alpha(||(\delta\mathbf{R} - \mathbf{I})\mathbf{t}^{pred} + \delta\mathbf{t}||_2^2) + u_\beta(||\delta\mathbf{R} - \mathbf{I}||_F^2), \tag{4}$$

to push $\mathbf{t}^{pred} \to \mathbf{t}^*$ and $\mathbf{R}^{pred} \to \mathbf{R}^*$, where $u_\diamond(\cdot)$ is a uncertainty-aware loss: $u_\diamond(l) = e^{-\diamond}l + \diamond$, to model the homoscedastic noise during training [35], where $\diamond$ is a learnable parameter. Ideally, the transformation residual $(\delta\mathbf{R}, \delta\mathbf{t})$ should be close to an identity transformation ($\mathbf{I}$, $\mathbf{0}$) after the training converges.

Deep Flow Supervision Loss. All the above losses are used to directly supervise the final ego-motion estimation. To further enhance the point-wise feature representations, we incorporate the 3D scene flow prediction as an auxiliary task in a unsupervised manner. The flow prediction highly relies on the local topology patterns, which could makes the features encode more geometric information. As shown in Fig. 5, we add several scene flow prediction branches (with several convolution layers) at the different depths of the ego-motion regression encoder to predict scene flow F^{pred} from the voxel features (each feature vector can be mapped to a 3D voxel location $\mathbf{x}_t^i$ with our voxel-based representation). The flow of a point $\mathbf{x}_t^i$ is defined as its coordinate difference between two timestamps: $F_h(\mathbf{x}_t^i) = \mathbf{x}_t^i - \mathbf{x}_{t-1}^i = (\mathbf{I} - \mathbf{R}^{gt})^{\mathrm{T}}\mathbf{x}_t^i + (\mathbf{R}^{gt})^{\mathrm{T}}\mathbf{t}^{gt}$. Since the ego-motion ground-truth is not available, we approximate the scene flow targets with the rectified ego-motion prediction with the transformation residuals obtained by the ICP iteration according to Eq. (3), and then calculate the scene flow of the voxels: $h \in \mathcal{H}$: $F_h^*(\mathbf{x}_t^i) = (\mathbf{I} - \mathbf{R}^*)^{\mathrm{T}}\mathbf{x}_t + (\mathbf{R}^*)^{\mathrm{T}}\mathbf{t}^*$, where $\mathbf{x}_t^i$ denotes the i-th voxel center's coordinate in encoder layer h. Finally, we take this approximated flow as the target to supervise the flow prediction at different encoder depths:

$$L_{fs} = \sum_{h,i} w_h \cdot u_\alpha(||F_h^{pred}(\mathbf{x}_t^i) - F_h^*(\mathbf{x}_t^i)||_2^2), \tag{5}$$

where w_h is the weight for layer h, and $u_\alpha(\cdot)$ is a uncertainty-aware loss mentioned before.

3.3 Correspondence Confidence Estimation

In practical scenarios, the nearest-neighbor-based correspondence mapping $\mathcal{M}(\cdot)$ used in the Eq. (2) and Eq. (4) to associate corresponding points in two timestamps are not always accurate, because of the existence of moving objects, noise, and measurement errors. To alleviate the adverse effects from these inaccurate correspondences, we design a 3D decoder sub-network following the geometric feature encoder to estimate the reliability for the points, which is implemented as 3D transposed convolution layers followed by sigmoid functions squeezing the output range to $[0, 1]$. As illustrated in Fig. 5, the confidence estimation decoder estimates the point-wise confidence $M'_t = \{m'^i_t | i = 1, \dots, N\}$ (trivially obtained from M_t) and $M_t = \{m^j_t | j = 1, \dots, N\}$ for point sets P'_t and P_t respectively. We takes pair-wise product $\{\mathrm{M}^{ij} = m'^i_t m^j_t | i = 1, \dots, N, j = \mathcal{M}(i)\}$ to estimate the reliability of each matched correspondence pair $(\mathbf{x}^i_t, \mathbf{x}'^j_t)$ needed by Eq. 2 and the ICP optimization for Eq. 4. The confidence factors can be straightforwardly incorporated into Eq. (2):

$$L_{sr} = -\frac{1}{|P_t|} \sum_{\mathbf{x}^i_t \in P_t, j = \mathcal{M}(i)} \frac{\mathrm{M}^{ij} \cdot \mathbf{x}^i_t \cdot \mathbf{x}'^j_t}{||\mathbf{x}^i_t||_2 ||\mathbf{x}'^j_t||_2}. \tag{6}$$

For the transformation residual loss (Eq. (4)), we modify the original ICP optimization term as

$$E = \frac{1}{|P_t|} \sum_{\mathbf{x}^i_t \in P_t, j = \mathcal{M}(i)} (\mathrm{M}^{ij} / \max_i(\mathrm{M}^{ij}) + \epsilon) \cdot \mathcal{D}(\mathbf{x}^i_t, \mathbf{x}'^j_t), \tag{7}$$

where $\mathcal{D}(\cdot, \cdot)$ denotes the distance function in ICP, $(\mathbf{x}^i_t, \mathbf{x}'^j_t)$ are the identified correspondences between P_t and P'_t with the nearest-neighbor mapping $\mathcal{M}(\cdot)$, and the constant addend ϵ (empirically set to 0.1 in our experiments) is to avoid the extreme imbalance of weights in early training phases. The classic ICP assumes that all correspondence pairs can be matched perfectly by optimizing the transformation ($\delta\mathbf{R}$,$\delta\mathbf{t}$) in Fig. 5b, which is often not true especially when moving objects/noises exist. The correspondence confidence factors successfully handle this dilemma to lower the weights on the unreliable correspondences during optimization.

Since there is no ground-truth for the confidence prediction, we use the self-supervised range alignment error to guide this confidence estimation similar to [8]:

$$L_{ra} = \frac{1}{|P_t|} \sum_{\mathbf{x}^i_t \in P_t, j = \mathcal{M}(i)} \mathrm{M}^{ij} (r(\mathbf{x}^i_t) - r(\mathbf{x}'^j_t))^2 - \gamma \log(\mathrm{M}^{ij}), \tag{8}$$

where $r(\cdot) = ||\cdot||_2$ calculates the range value and the regularization term $-\log(\mathrm{M}^{ij})$ with weight γ avoids the all-zero trivial prediction.

In summary, our overall loss function is finally expressed as

$$L = w_1 L_{sr} + w_2 L_{ra} + w_3 L_{tr} + w_4 L_{fs}, \tag{9}$$

where w_1-w_4 are the weights of different losses.

4 Experimental Results

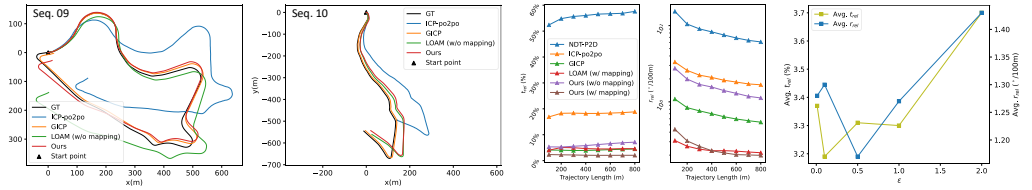
4.1 Benchmark Dataset and Evaluation Metrics

KITTI Odometry Dataset. KITTI Odometry dataset[12] consists of 22 LiDAR sequences with corresponding color/gray images. It provides ground-truth poses for sequences 00-10 obtained from IMU/GPS fusion algorithms. The remaining sequences are for benchmark testing and do not provide ground-truth poses. This dataset covers different types of road environments (including country roads, urban areas, highways *etc*.), and contains pedestrains, cyclists and different types of vehicles. The speed of acquisition vehicles varies in different areas ranging from 0 km/h to 90 km/h.
Apollo-SouthBay Dataset. Apollo-SouthBay Dataset [13] collected in the San Francisco Bay area, United States, covers various scenarios including residential area, urban downtown areas and highways. It also provides the ground-truth poses and training/testing splits for the first five scenarios which are adopted in our experiments. The apollo sequence is longer than the KITTI's and the scenarios are more complicated, which is suitable for testing the generality of our method.
We adopt the official evaluation metrics provide by the KITTI benchmark [12] to measure the translational and rotational drift on length of 100m-800m in our experiments. The implementation details of our method can be found in supplementary materials.

Table 1: Comparison with the state of the arts on KITTI odometry dataset. We also list the supervised methods and visual odometry methods here for reference.

	Seq.	Training Seq.		7		8		9		10		Testing Avg.	
		t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
Classic	ICP-po2po	6.45	3.16	5.17	3.35	10.04	4.93	6.93	2.89	8.91	4.47	7.76	3.98
	ICP-po2pl	3.76	1.79	1.55	1.42	4.42	2.14	3.95	1.71	6.13	2.60	4.01	1.97
	GICP [2]	1.87	0.76	0.64	0.45	1.58	0.75	1.97	0.77	1.31	0.62	1.38	0.65
	NDT-P2D [36]	34.2	5.73	7.51	3.07	13.6	4.62	33.7	7.06	20.5	3.06	18.8	4.45
	CLS [37]	2.30	0.93	1.04	0.73	2.14	1.05	1.95	0.92	3.46	1.28	2.15	1.00
	LOAM (w/o mapping)	3.90	1.53	2.98	1.55	4.89	2.04	6.04	1.79	3.65	1.55	4.39	1.73
	LOAM (w/ mapping) [4]	1.12	0.50	0.69	0.50	1.18	0.44	1.20	0.48	1.51	0.57	1.15	0.50
Sup.	Velas *et al.* [9]	2.90	-	1.77	-	2.89	-	4.94	-	3.27	-	3.22	-
	LO-Net (w/o mapping) [10]	1.05	0.66	1.70	0.89	2.12	0.77	1.37	0.58	1.80	0.93	1.75	0.79
	LO-Net (w/ mapping) [10]	0.71	0.45	0.56	0.45	1.08	0.43	0.77	0.38	0.92	0.41	0.83	0.42
Unsup.	Zhou *et al.* * [8]	30.8	4.63	21.3	6.65	21.9	2.91	18.8	3.21	14.3	3.30	19.1	4.02
	UnDeepVO* [18]	4.80	2.69	3.15	2.48	4.08	1.79	7.01	3.61	10.6	4.65	6.22	3.13
	Cho *et al.* [11]	3.68	0.87	-	-	-	-	4.87	1.95	5.02	1.83	4.95	1.89
	Ours	2.50	1.11	3.09	1.81	3.16	1.14	3.01	1.14	3.48	1.11	3.19	1.30
	Ours (more data)	2.31	1.06	2.51	1.15	2.65	1.00	2.86	1.17	3.22	1.26	2.81	1.15
	Ours (more data, w/mapping)	**0.70**	**0.37**	**0.34**	**0.21**	1.18	**0.35**	**0.83**	**0.34**	**1.22**	**0.40**	**0.89**	**0.32**

t_{rel}, r_{rel}: Average translational RMSE (%) and rotational RMSE ($^\circ$/100m) on length of 100m-800m [12]. *: Visual odometry methods. Some results were obtained from [10, 11]



(a) (b) (c)

Figure 3: (a) Trajectory plots comparison among some two-frame methods on KITTI. (b) The average translation and rotation errors with respect to trajectory length intervals for different methods on Apollo. (c) Performance variation of our method on KITTI test set with different ϵ values.

4.2 Comparison with State-of-the-arts

Evaluation on KITTI Dataset. We compare our method with other competitive methods on the sequences 00-10 of the KITTI odometry dataset. These compared methods adopted different splitting strategies for training and testing [1]. *Since most of their code are not available, for fair comparison, we choose the splitting strategy with minimal training data (i.e. 00-06/07-10 for training/testing) to evaluate our method against them.* Table 1 shows the evaluation results where we exclude the results of Seq. 01 following other unsupervised methods [11]. The Seq. 01 on highway is in a very open space with few structures to infer the agent's motions, where most of the unsupervised method and classic methods fail. Compared with other state-of-the-art unsupervised methods, our method achieves the best performance (denoted as 'Ours' in Table 1) even with the least amount of data for training. We also try to add the more unlabeled (Seq. 11-21) into the training set (denoted as 'Ours (more data)'), and our performances are further improved and even surpasses those of some supervised methods, which shows our unsupervised method's scalability to be benefited from more training data. We also compare our method with classic methods [1, 36, 2, 37]. Our method achieves significant better performance than most two-frame methods widely used, *i.e.*, point-to-point ICP [1], point-to-plane ICP [33] and NDT-P2D [36], and reasonably inferior performance than the time-consuming (only ~ 0.5Hz) GICP [2] and CLS [37] which iteratively refine the pose transformation with computational heavily correspondence association modeling. (Specific runtime comparison can be found in the supplementary materials.) Fig. 3a plots example trajectories in Seq. 09-10 of different two-frame methods for visualization. Although we mainly focus on the two-frame ego-motion estimation in this paper, we also incorporate the mapping module from LOAM [4] into our framework to test the odometry performance with the additional backend multi-frame refinement. The results (denoted as 'Ours (more data, w/ mapping)') in Table 1 demonstrate that our method can be successfully coupled with multi-frame refinement widely used in SLAM systems. We achieves better performance than LOAM, a competitive classic LiDAR-based SLAM system, and show comparable performance with state-of-the-art supervised odometry method LO-net [10] with mapping refinement.

[1] [9] takes the sequences 00-07/08-10 as the train/test split, [10] takes 00-06/07-10 as the train/test sets, while the others CNN-based methods takes 00-08/09-10 for training/testing

Table 2: The evaluation results on Apollo-SouthBay test set.

	ICP-po2po		ICP-po2pl		GICP [2]		NDT-P2D [36]		LOAM(w/ mapping) [4]		Ours		Ours (w/ mapping)	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
Avg.	22.8	2.35	7.75	1.20	4.55	0.76	57.2	9.40	5.93	0.26	6.42	1.65	**2.25**	**0.25**

Table 3: Comparison among different ablation variants.

	L_{sr}		L_{sr}, L_{ra}		L_{sr}, L_{ra}, L_{tr}		$L_{sr}, L_{ra}, L_{tr}, L_{fs}, w/o\ conf.$		$L_{eu}, L_{ra}, L_{tr}, L_{fs}$		$L_{sr}, L_{ra}, L_{tr}, L_{fs}$	
	train	test	train	test	train	test	train	test	train	test	train	test
Avg. t_{rel}	12.4	19.1	4.63	7.19	2.59	4.84	3.24	3.59	2.88	3.50	**2.50**	**3.19**
Avg. r_{rel}	4.72	6.54	1.90	2.65	1.20	1.96	1.82	2.04	1.41	1.27	**1.11**	**1.30**

Evaluation on Apollo-SouthBay Dataset. We also evaluate our model on more challenging Apollo-SouthBay dataset to further demonstrate our generality. As shown in Table 2 and Fig. 3b, our two-frame-based network prediction (denoted as 'Ours') consistently outperform most of the classic two-frame-based methods [1, 33, 36] and achieves comparable translational accuracy to the multi-frame-based LOAM [4]. We also prove that our method with mapping refinement can outperform other methods. The Apollo dataset is closer to actual autonomous driving scenarios with more moving objects. By training on large-scale data and correspondence confidence estimation, our method shows consistent robustness.

4.3 Ablation Study

To verify the effectiveness of each proposed module and unsupervised loss functions, we conduct a throughout ablation study on the KITTI dataset as shown in Table 3. We test different combinations of loss functions incrementally and the proposed full loss to show that it has optimal performance compared with losses with fewer terms. We also try to remove our confidence mechanism from our model (denoted as 'w/o conf.'), and we can see evident performance drop due to unreliable correspondences caused by noises, dynamic objects and varying pointcloud densities. We visualize some estimated confidence examples in Fig. 4, where our network successfully lowers the confidences of points on dynamic objects, *e.g.* vehicles, cyclists *etc.* and has higher confidence on the static poles and vertical surfaces. Moreover, we substitute the Euclidean loss L_{eu} for our spherical reprojection loss L_{sr} and find obvious performance drop. Besides, in Fig. 3c, we analyse the the performance variations with different ϵ values in Eq. 7 and find that large ϵ values lead to inferior performances, as the large ϵ weakens the effect of estimated confidence weights.

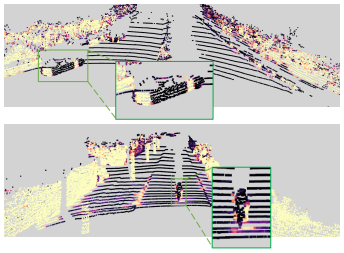


Figure 4: Our confidence prediction visualization. The brighter denotes higher confidendence.

Table 4: The runtime of our submodules and the comparison with other methods.

Our submodule runtime	
Module	Time (ms)
Voxelization	0.7
Voxel Feature Extraction	65.8
Odometry Regression	21.9
Total	88.4
Runtime comparison with other methods	
Methods	Time (ms)
ICP-po2po [1]	261
ICP-po2pl [33]	1250
GICP [2]	1781
NDT-P2D [36]	1723
CLS [37]	19843
Ours	88.4

4.4 Runtime Analysis

We further analyse the running time of our framework on a machine with a Xeon(R) E5-2697A v4 CPU and a NVIDIA Tesla V100 GPU. The results are listed in Table 4. Our method achieve real-time efficiency, which is suitable for practical deployment. Although GICP can achieve more robust results than our network prediction in Table 1 and Table 2, it is too time-consuming to be directly applied to large-scale LiDAR data.

5 Conclusions

We present a novel unsupervised LiDAR odometry framework based on the 3D convolutional neural networks. Several 3D self-supervised losses are proposed and uncertainty-aware mechanism is introduced to jointly enable our network to train in the wild. Our method achieve the state-of-art performance on two public datasets. It is worth mentioning that our method can run in real-time and can be combined with other off-the-shelf mapping algorithms for deployment in practical applications.

Acknowledgments

This work is supported in part by the General Research Fund through the Research Grants Council of Hong Kong under Grants(*Nos.* CUHK14208417, CUHK14207319), in part by the Hong Kong Innovation and Technology Support Program (*No.* ITS/312/18FX), in part by CUHK Strategic Fund.

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[2] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[3] J. Serafin and G. Grisetti. Nicp: Dense normal based point cloud registration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 742–749, 2015.

[4] J. Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

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

[6] K. R. Konda and R. Memisevic. Learning visual odometry with a convolutional network. In *VISAPP (1)*, pages 486–490, 2015.

[7] S. Wang, R. Clark, H. Wen, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050, 2017.

[8] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[9] M. Velas, M. Spanel, M. Hradis, and A. Herout. Cnn for imu assisted odometry estimation using velodyne lidar. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 71–77, 2018.

[10] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li. Lo-net: Deep real-time lidar odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019.

[11] Y. Cho, G. Kim, and A. Kim. Unsupervised geometry-aware deep lidar odometry. In *2020 International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[13] W. Lu, Y. Zhou, G. Wan, S. Hou, and S. Song. L3-net: Towards learning based lidar localization for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6389–6398, 2019.

[14] E. W. Weisstein. Angular distance. URL https://mathworld.wolfram.com/AngularDistance.html.

[15] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE, 2001.

[16] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga. Fast registration based on noisy planes with unknown correspondences for 3-D mapping. *IEEE Transactions on Robotics*, 26(3):424–441, 2010.

[17] J. Behley and C. Stachniss. Efficient surfel-based SLAM using 3D laser range data in urban environments. In *Robotics: Science and Systems*, 2018.

[18] R. Li, S. Wang, Z. Long, and D. Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7286–7291, 2018.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[21] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3D object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.

[22] X. Gu, Y. Wang, C. Wu, Y. J. Lee, and P. Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3254–3263, 2019.

[23] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[24] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18 (10):3337, 2018.

[25] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[26] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3D object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[27] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar++: Accurate depth for 3D object detection in autonomous driving. *arXiv preprint arXiv:1906.06310*, 2019.

[28] X. Liu, C. R. Qi, and L. J. Guibas. Flownet3D: Learning scene flow in 3D point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019.

[29] S. Prokudin, C. Lassner, and J. Romero. Efficient learning on point clouds with basis point sets. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[30] B. Graham and L. van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[33] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[34] K.-L. Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004.

[35] A. Kendall and R. Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017.

[36] T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3D ndt representations. *The International Journal of Robotics Research*, 31(12):1377–1393, 2012.

[37] M. Velas, M. Spanel, and A. Herout. Collar line segments for fast odometry estimation from velodyne point clouds. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4486–4495, 2016.

[38] B. L. Yan Yan. Spconv: Pytorch spatially sparse convolution library, 2020. URL https://github.com/traveller59/spconv.

6 Appendix

6.1 Network Details

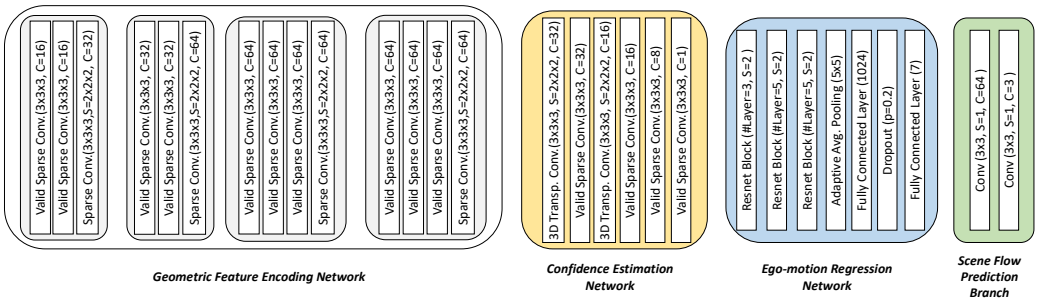


Figure 5: The design details of our proposed modules.

As shown in Fig. 5, the geometric feature encoding network adopts interlaced valid sparse convolutions and sparse convolutions proposed by Graham *et al*. [30][2] to encode the point clouds into high-dimension features. The confidence estimation network is constituted by interlaced 3D transposed convolutions and valid sparse convolutions to upsample and decode the confidences from the high-level features generated by the geometric feature encoding network. Moreover, the specific design of ego-motion regression network and the scene flow prediction branch are also presented in Fig. 5. Note that most of the convolution layers are followed by a ReLU layer and a batch normalization layer except for those in the geometric feature encoding network where we omit the batch normalizations and find better performance.

6.2 Implementation Details

Our proposed network is implemented with PyTorch and trained on the NVIDIA V100 Tesla V100 GPU. We voxelize the 3D space of size (D=137.6m, W=80m, H=8m) into small voxels of (0.1m,0.1m,0.2m). We set the initial learning rate to 0.001, which slowly decays with a cosine annealing strategy. The batchsize is set to 8 and the network is trained 200k iterations to achieve good convergence. During training, we set the length of input sequences to 3 and form the temporal pairs by choosing scans $[S_{t-2}, S_{t-1}]$, $[S_{t-1}, S_t]$, $[S_{t-2}, S_t]$ respectively. We exponentially decay the w_h in Eq. (5) and set $w_h = 1, 0.1, 0.01 \ldots$ (indexed from the end of the encoder). Besides we set $\epsilon = 0.1$, $\gamma = 0.0001$, $w_1 = 1000$, $w_2 = w_3 = w_4 = 1$ in Eq. (7), Eq. (8) and Eq. (9) as the base settings in our experiment.

[2]The valid sparse convolution, sparse convolution and the following transposed convolution in our work are implemented with [38]