Lidar Mapping Optimization Based on Lightweight Semantic Segmentation

Zhihao Zhao, Wenquan Zhang, Jianfeng Gu, Junjie Yang, and Kai Huang

***Abstract*—Lidar Simultaneous Localization and Mapping (LiDAR-SLAM) algorithm with semantic information is an open research question and it is such a time consuming task. The related work that focus on real-time LiDAR-SLAM has poor accuracy. To solve these problems, a lightweight semantic segmentation network to assist in localization and mapping is proposed in this paper. The method uses the lidar point clouds generated by the simulator and annotated manually in real world as the original input. Then, the semantic cloud is segmented by the semantic segmentation network to obtain the semantic information. Finally, the semantic information obtained by the segmentation is used to assist the localization and mapping.**

***Index Terms*—Lightweight network, semantic segmentation, semantic mapping, SLAM, point cloud generator.**

I. Introduction

LIDAR Simultaneous Localization and Mapping (LiDAR-SLAM) refers to obtain the map information based on lidar point clouds. LiDAR-SLAM is common as lidars can provide high frequency range measurements where errors are relatively constant irrespective of the distances measured. In terms of distance measurement, the lidar sensor not only provides centimeter-level distance information, but also has a wide detection range. Autonomous robots mainly use lidar sensors for simultaneous localization and mapping because of the better illumination robustness and higher distance measurement accuracy.

There are many LiDAR-SLAM algorithm have been proposed [1]–[4] to get stable and precise map information. However, these LiDAR-SLAM algorithm are such a time consuming task when dealing with the heavy point clouds and computing platform cannot provide enough power to deploy on an embedded platform as automotive industry is particularly sensitive in terms of hardware costs [5], [6] and [7] presents a case study to test the feasibility. Therefore, it cannot be deployed to commercial vehicles. This leads to the requirement that the system with higher accuracy should also have a fast LiDAR-SLAM algorithm at the same time.

LOAM [8] is a real-time LiDAR-SLAM algorithm. The key idea of LOAM is to divide SLAM into two part. One part performs odometry at a high frequency but low fidelity to estimate velocity of the lidar. Another part runs at a frequency of an order of magnitude lower for fine matching and registration of the point cloud. Combination of the two part allows the method to map in real-time. LOAM has a good performance in real time and motion estimation between frames, but the accuracy of localization is susceptible to feature points. LOAM use the curvature of the point clouds as the feature. When the pedestrian and the vehicle are moving between two frames, the feature points will fall on the moving objects. The distance of feature points between the pedestrian and the vehicle will increase, and then the displacement between frames calculated according to the feature point matching relationship will cause a large error. So it is important to remove the moving feature point and obtain the good feature point at the same time. Semantic information is a effective way to solve the problem. Therefore, in order to improve accuracy of the real-time LiDAR-SLAM, a fast and lightweight semantic segmentation network should be proposed to provide semantic information.

Semantic segmentation based on convolutional neural network (CNN) has been proposed as an end-to-end solution for LiDAR point cloud feature extraction and classification. As the LiDAR point cloud is sparse due to the non-uniform sampling of the 3D space, there are two basic approaches for processing the sparse LiDAR point cloud in CNN: 1) 3D CNN is applied by encoding point cloud in 3D voxels [9]–[11], and 2) 2D image-style CNN is applied with point cloud projected into dense matrix [12]–[14]. For those sparse and unordered points in LiDAR scans, 2D dense matrix is more memory-efficient at point-wise resolution compared to 3D voxels. However, in order to extract features within images, such 2D CNN model is designed with huge amount of convolution layers and convolution filters. It is too heavy for LiDAR point cloud processing in terms of model size and model complexity.

In view of the aforementioned problems, this paper proposes a method based on lightweight semantic segmentation to assist in localization and mapping. This method uses the lidar point clouds generated by the simulator and annotated manually in real world as the original input. Then, the semantic cloud is segmented by the semantic segmentation network to obtain the semantic information. Finally, the semantic information obtained

Manuscript received September 18, 2018; revised January 17, 2019; accepted March 29, 2019. Date of publication May 28, 2019; date of current version August 23, 2019. This work was supported by the National Natural Science Foundation of China under Grant 67000-41030095. *(Corresponding author: Kai Huang.)*

Z. Zhao, J. Gu, J. Yang, and K. Huang are with the Key Laboratory of Machine Intelligence and Advanced Computing, Ministry of Education, School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510245, China (e-mail: zhaozhh25@mail2.sysu.edu.cn; gujf5@mail2.sysu.edu.cn; yangjj27@mail2.sysu.edu.cn; huangk36@mail.sysu.edu.cn).

W. Zhang is with the UISEE Technology, Inc., ShenZhen 518000, China (e-mail: zhangwq35@mail2.sysu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIV.2019.2919432

by the segmentation is used to assist the localization and mapping. The contributions of this paper are summarized as follows:

- A lightweight, fast and accurate LiSeg for LiDAR point cloud semantic segmentation based on depth-wise separable convolution is proposed. The model size of LiSeg is at least eight times smaller than the related work in the literature, while still achieving high accuracy.
- We integrate semantic information into LiDAR-SLAM and then generate the semantic map through three processes:feature point selection, feature point matching selection between frames by KNN(K-Nearest Neighbor), and motion trajectory removal.
- We perform lots of experiments. Semantic segmentation result shows that the proposed lightweight LiSeg network still achieves mean-IoU of 0.72 and 0.71 on KITTI data set and our Velodyne HDL-32E data set, respectively. The LiSeg performs point cloud semantic segmentation at 111 frames per second(fps) on a GTX 1070 GPU. Mapping result shows this work get approximately 35% improvement of the localization accuracy in the traffic-stricken sections.

The rest of this paper is organized as follows: Section II presents the related work and how our work is outstanding from the state of the art. Section III presents what the data generator is and how to get annotated point clouds from Carla simulator, then proposes a lightweight, fast and accurate FCN LiSeg for LiDAR point cloud semantic segmentation. After that, the method will optimize the generation of map using semantic information. Section IV shows the experiments results. Section V concludes the whole work.

II. Related Work

In the process of constructing semantic maps, it is necessary to correlate geometric information and semantic information in the environment through object recognition and classification. In addition to information association based on traditional classification models [15],with the development of research in recent years, some researchers have combined simultaneous location and mapping with deep learning to correlate the geometric and semantic information in the environment, thus generating semantic maps.

In the field of simultaneous localization and mapping, there are mainly two kinds of methods: visual simultaneous localization and mappingV-SLAM), lidar simultaneous localization and mapping(LiDAR-SLAM), which depend on camera and LiDAR sensor respectively. The Monocular Camera cannot provide 3D distance information. The Stereo Camera and the RGB-D Camera have a detection range of about 20 meters, and the accuracy of the depth information is inversely proportional to the detection range [16]. In terms of distance measurement, the lidar sensor not only provides centimeter-level distance information, but also has a wide detection range. For example, on the well-known test benchmark KITTI Benchmark [17], the algorithm with the highest localization accuracy and the second highest are dependent on the lidar sensor, as shown in the following Table I.[1] Among them, the algorithm of fusion laser-visual odometry V-LOAM [3] has achieved the most accurate results with translation error 0.62% and rotation error (deg/m) 0.0014. Algorithms using only lidar sensors include LOAM [18] and IMLS-SLAM [4]. SOFT2 [19], RotRocc+ [20], GDVO [21] and ORB-SLAM2 [22] based on stereo cameras still have a certain gap in accuracy. Because lidar sensors have better illumination robustness and higher distance measurement accuracy, autonomous robots mainly use lidar sensors for simultaneous localization and mapping.

TABLE I
PRECISION COMPARISON OF CURRENT STATE-OF-THE-ART SLAM ALGORITHMS

algorithm	translation error	rotation error (deg/m)	sensor type
V-LOAM[3]	0.62%	0.0014	cameralidar
LOAM[3]	0.64%	0.0014	lidar
SOFT2[17]	0.65%	0.0014	stereo camera
IMLS-SLAM[4]	0.69%	0.0018	lidar
RotRocc+[18]	0.83%	0.0026	stereo camera
GDVO[19]	0.86%	0.0031	stereo camera
ORB-SLAM2[20]	1.15%	0.0027	stereo camera

In the work of Dewan [23], the FasetNet model proposed by Oliveira *et al.* [24] was used to segment the road image for point cloud semantic segmentation. In Wu *et al.* [14], SqueezeSeg model proposed by Nanfack *et al.* [25] was used for image semantic segmentation. However, the above model is mainly designed for image semantic segmentation. The model has more layers and a higher number of convolution kernels. Training and inference are time consuming and not suitable for real-time.

In terms of environmental semantic information association, Hackel [11] *et al.* generate semantic maps by semantic segmentation of lidar point cloud maps based on environmental geometric information. Xiao [26] and Lai [27] *et al.* constructed semantic point cloud maps by semantically segmenting RGB-D images and using semantic segmentation results to perform 3D reconstruction. Yang *et al.* [28] proposed the Pop-up SLAM method, which uses the method of plane door frame model matching to identify the position of the door frame in the map and to generate a semantic map. Hua *et al.* [29] segmented the point cloud grid and merged it using Markov fields to provide semantic annotation of the 3D mesh in combination with manual correction. Bowman *et al.* [30] combines object detection in images and V-SLAM to identify the location of the detected object in the map and generate a semantic map. Yang *et al.* [31] use the conditional random field to segment the image semantically and use the above segmentation image to generate a semantic map. Xiang *et al.* [32] proposed the DA-RNN algorithm to construct a semantic map by using recursive neural networks to correlate map and image semantic information. McCormac *et al.* [33] and Gao *et al.* [34]proposed the Semantic Fusion method to generate a semantic map, which uses a CNN to semantically segment the image.

In view of the problems aforementioned, we propose a method based on lightweight semantic segmentation to assist in localization and mapping. Firstly, the method uses the lidar point clouds generated by the simulator and annotated manually in real world as the original input. Then, the semantic segmentation network LiSeg segment the point clouds to obtain the semantic information. Finally, the semantic information is used in LOAM-SLAM

[1]data from: http ://www.cvlibs.net/datasets/kitti/eval_odometry.php

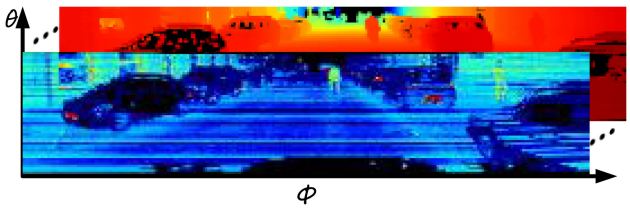


Fig. 1. Dense matrix in Spherical coordinates.

to remove moving object and assist the localization and mapping.

III. Approach

Our approach include two parts, semantic segmentation network and point clouds mapping. We propose a lightweight semantic segmentation network to assist in localization and mapping is proposed in this paper. The method uses the lidar point clouds generated by the simulator and annotated manually in real world as the original input. Then, the semantic cloud is segmented by the semantic segmentation network to obtain the semantic information. Finally, the semantic information obtained by the segmentation is used to assist the localization and mapping.

A. Liseg Network Structure

1) Data Pre-Processing: The point cloud in LiDAR scans is sparse, and point density varies a lot due to the distance and shape of objects. In order to process the sparse and unorganized LiDAR point cloud with deep convolution neural network, the point cloud is encoded in dense matrix in 2D format. The point cloud is transformed into Spherical coordinates $\mathcal{S}(\theta, \phi)$, as shown in Fig. 1, from Cartesian coordinates $\mathcal{C}(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$. Other properties of points are encoded as channels of the dense matrix. The shape of the matrix is thus defined as $[H, W, C]$, which represents [θ, ϕ, $channels$] respectively. H is the number of LiDAR scan lines. W is set according to the horizontal resolution($\tilde{\phi}$) of LiDAR. C is set according to the number of channels required.

For each $\mathcal{P}_i$ in Cartesian coordinates point cloud $\mathcal{P}$, the corresponding transformation into Spherical coordinates $\mathcal{S}(\theta_i, \phi_i)$ is shown as Eq.(1) and Eq.(2). In this case, $\mathcal{RING}(\mathcal{P}_i)$ is the scan line number of $\mathcal{P}_i$, which can be achieved directly from the Velodyne driver.

$$\theta_i = \mathcal{RING}(\mathcal{P}_i) \tag{1}$$

$$\phi_i = \arcsin \frac{\mathcal{P}_i.y}{\sqrt{\mathcal{P}_i.x^2 + \mathcal{P}_i.y^2}}, \tilde{\phi} = 0.175^\circ \tag{2}$$

Here, the proposed Liseg network is trained on KITTI data set, which used a Velodyne HDL-64E with 64 scan lines, 10 Hz rotations frequency and 0.175° horizontal resolution($\tilde{\phi}$). In order to enhance memory efficiency and generalization of representation, C here is set to 5, with only 5 channels: intensity, x, y, z, and range ($range = \sqrt{x^2 + y^2 + z^2}$). Such channels are the raw properties of LiDAR point clouds, which is enough to present the distribution of points within 3D space. In KITTI data set, only 90° of field of view is annotated. Thus, in this case, the H is set to 64 for KITTI data set and 32 for Velodyne HDL-32E data set, W is set to $90/0.175 = 512$, and C is set to 5.

2) Network Structure: In order to be deployed on mobile devices for autonomous driving, Liseg is designed as a lightweight model according to the common encoder-decoder FCN style but with separable convolution and dilated convolution. The separable convolution performs a depth-wise convolution that acts separately on channels, followed by a point-wise convolution that mixes channel-wise features. In this case, depth-wise separable convolution is used for reducing parameters. In addition, due to the great differences of the distribution of intensity, x, y, z, and range, depth-wise convolution is more suitable. Dilated convolution is used in the encoder for enhancing the reception field. The whole Liseg network encompasses only convolutional and deconvolutional layers followed by Batch Normalization(BN) and ReLu(RL) non-linearities.

The structure of Liseg is shown in Fig. 2. Depth-wise separable convolution layer is shown in blue. Max-pooling layer is shown in green. Dilated convolution layer is shown in yellow. Transpose convolution layer is shown in grey. Convolution is shown in black. The convolution operators are parameterized as $(F, K1, K2, S1, S2)$ and the max-pooling operators are parameterized as $(K1, K2, S1, S2)$, where F is the number of filters, $(K1, K2)$ is the kernel size respectively, and $(S1, S2)$ is the strides respectively except for dilated convolution layers. For dilated convolution layers, $(S1, S2)$ stands for the dilated rate.

As Liseg is designed in lightweight manner, two depth-wise separable convolution layers are applied for reducing parameters and computation [35]. In order to reduce parameters of feature maps, and improve timing performance, two max pooling layers are used in early stage for sampling the feature maps by a half. Three dilated convolution layers with dilated rate: (1, 2), (1, 4) and (1, 2) are used for feature extraction. Skip-connection techniques are used in the decoding step by providing features in early stages as enhance-concatenate component. After two transpose convolution layers, the final score map is generated by a convolution layer. For the first depth-wise separable convolution layer with kernel size (3, 3), the number of input channels $\mathcal{C}_{in1}$ is 5, and number of output channels $\mathcal{C}_{out1}$ is 20. For the second depth-wise convolution layer with kernel size (3, 3), $\mathcal{C}_{in2}$ is 20 and $\mathcal{C}_{out2}$ is 32. After applying depth-wise separable convolution, the number of parameters in the first layer is reduced from 900($\mathcal{C}_{in1} * \mathcal{C}_{out1} * 3 * 3$) to 149($\mathcal{C}_{in1} * 3 * 3 + \mathcal{C}_{in1} * \mathcal{C}_{out1} * 1 * 1$) by 84%. The number of parameters in the second layer is reduced from 5760 to 820 by 86% in the same manner.

However, due to the low resolution of the dense matrix and the imbalance of H and W, e.g. ($H = 64$, $W = 512$) and ($H = 32$, $W = 512$), sub-sampling is only carried out on columns W within the encoder module.

Such three layers provide multi-scale features with reception field: 3*7, 3*15, and 5*11. In this case, spatial drop-out layers are applied for the above three dilated convolution layers, while training with Velodyne HDL-32E data set. The multi-scale feature maps from the three dilated convolution layers are then concatenated as the input of decoder for transpose convolution,

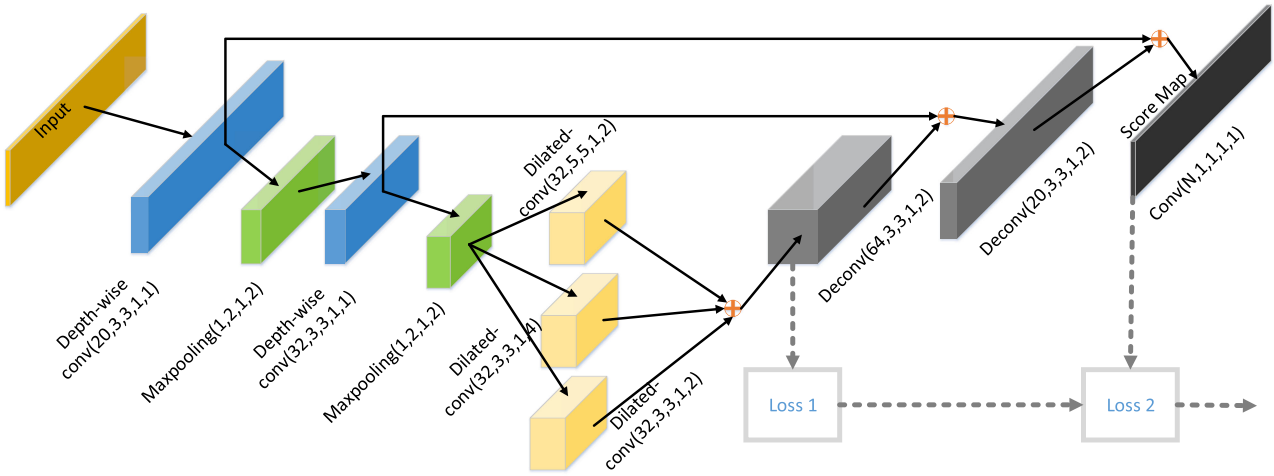


Fig. 2. LiSeg Structure. Depth-wise separable convolution convolution is shown in blue. Max-pooling layer is in green. Dilated is shown in yellow. Transpose convolution is shown in grey. Convolution is shown in black.

similar to Inception module but dilated convolution is used rather than convolution.

The whole LiSeg network is trained via end-to-end back-propagation guided by the following weighted multi-layer loss function:

$$\mathcal{L} = \sum_{i=1}^{2} \lambda_i \mathcal{L}oss_i{}^{WCE}(\hat{\mathcal{Y}}_j, \mathcal{Y}_j) \tag{3}$$

The above weighted multi-layer loss $\mathcal{L}$ is evaluated on two feature maps respectively, regarded as $\mathcal{L}oss_1$ and $\mathcal{L}oss_2$, as shown in Fig. 2. The hyper parameter λ_i is the corresponding regularization weights. $\hat{\mathcal{Y}}$ and $\mathcal{Y}$ are the predictions and ground truth respectively.

The number of points in the LiDAR point cloud are class-imbalance, because most of points belong to background. Thus the following multi-class Weighted Cross Entropy(WCE) loss is applied as regularization, because of the class-imbalance problem of the LiDAR point cloud. The corresponding class-imbalance regularization weights are computed according to the training set statistics.

$$\mathcal{L}oss_i^{WCE} = -\sum_{j,k,l}^{H_i,W_i,N} \mathcal{W}(\mathcal{Y}_{j,k}) \mathcal{ID}(\mathcal{Y}_{j,k}) \log(\hat{\mathcal{Y}}_{j,k,l}) \tag{4}$$

The $\mathcal{ID}(\mathcal{Y}_{j,k})$ is an index function to select the expected classes, and $\mathcal{W}(\mathcal{Y}_{j,k})$ is corresponding the class-imbalance weights.

B. Point Clouds Mapping and Optimization

1) Lidar Odometry and Mapping: Lidar Mapping includes three sub-modules : point cloud registration, lidar odometry and mapping.

Point cloud registration make preliminary transformation of the point position in a frame through the preliminary position estimation. lidar odometry extracts the feature points of the point cloud frame, and then make transformation between frames with feature point matching relationship.After the transformation between frames is obtained, the mapping sub-module performs coordinate transformation on the point cloud and generates a point cloud map.

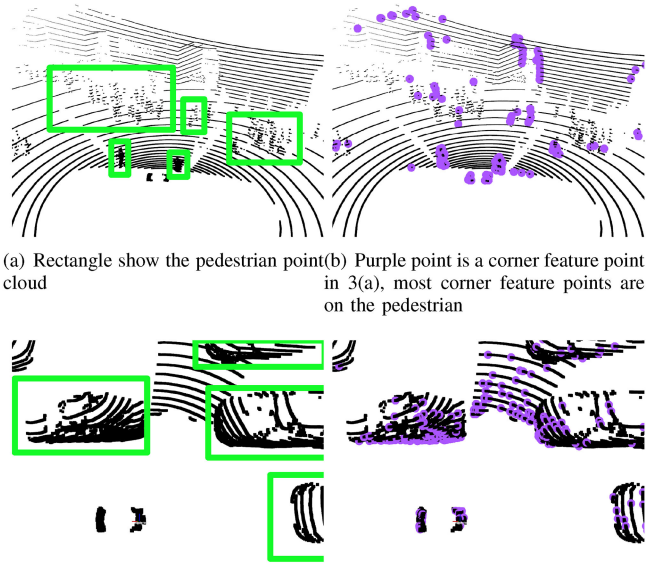
$$C = \frac{1}{|S| \, \|P_{(k,i)}\|} \| \sum_{j \in S, j \neq i} (P_{(k,i)} - P_{(k,j)}) \| \tag{5}$$

Lidar odometry use the curvature of the point clouds as the feature. The curvature's formula is shown in 5. Where $P_{(k,i)}$ represents the ith point of the kth frame point cloud, and S is the point produced by the same laser with $P_{(k,i)}$. The corner feature points are higher than a certain threshold, and the plane feature points are below the threshold. The adjacent corner feature points between frames should be on a straight line with constraint condition of the real world and the adjacent plane feature points should be on a plane.

$$d_\varepsilon = \frac{|(\widetilde{P}_{(k,i)} - \overline{P}_{(k-1,j)}) \times (\widetilde{P}_{(k,i)} - \overline{P}_{(k-1,l)})|}{|(\overline{P}_{(k-1,j)} - \overline{P}_{(k-1,l)})|} \tag{6}$$

$$d_H = \frac{\left| (\widetilde{P}_{(k,i)} - \overline{P}_{(k-1,j)}) \right|}{|(\overline{P}_{(k-1,j)} - \overline{P}_{(k-1,l)}) \times (\overline{P}_{(k-1,i)} - \overline{P}_{(k-1,j)})|} \tag{7}$$

where $\widetilde{P}_{(k)}$ is the feature point of the current frame, and $\overline{P}_{(k-1)}$ is the feature point of the previous frame. As shown in the formula (6), the transformation of the point cloud between frame should make the distance d_ε minimum between the corner feature point of the current frame and the neighbor frame. The plane point cloud has the distance shown in the formula (7),that should also be minimum between current frame and neighbor frame. The detailed derivation can be seen in [8] . The Lidar odometry uses Levenberg-Marquardt (LM) to minimize d_ε and d_H to register point clouds between frame and then estimate the movement between frame. After the transformation, point cloud mapping is obtained, the coordinate transformation of the point cloud frame is performed and then a point cloud map is generated.



(a) Rectangle show the pedestrian point cloud

(b) Purple point is a corner feature point in 3(a), most corner feature points are on the pedestrian

(c) Rectangle show the car point cloud

(d) Purple point is a plane feature point in 3(c), most plane feature points are on the flat surface of the car

Fig. 3. Point cloud corner and plane feature point extraction.

Algorithm 1: Feature Point Selection Pseudocode.

Input: Pf : Point clouds extracted preliminary
Result: Pf_c : Candidate feature point

```
Main Loop:
for k = 1 : |Pf| do
    if p_k.class is not movable then
        Pf_c ⟵ p_k ;
    end if
end for
return Pf_c
```

2) Feature Point Selection Optimization: Lidar odometry and mapping algorithm extracts the corner feature points and the plane feature points through curvature which is the geometric feature of the point cloud. However, the corner feature points and the plane feature points in some point clouds belong to objects that are belong to moving objects, and cannot exist stably in a fixed position in the map between multiple frames. Taking the autonomous car as an example, as shown in Fig. 3, there are a lot of corner feature points in the pedestrians as shown in Fig. 3(a) and (b), and there are also a lot of plane feature points in the vehicles as shown in Fig. 3(c) and (d). When the pedestrian and the vehicle are moving between two frames, the distance between the pedestrian and the corresponding feature points of the vehicle will increase, and the displacement between frames which calculated according to the feature point matching relationship will cause a large error.

To solve this problem, this paper select the feature points based on curvature extraction through semantic information. Taking a autonomous car as an example, pedestrians and vehicles in the scene are more likely to move. Therefore, the selection of the feature points should avoid the objects.The following Algorithm 1 is used to select and optimize the feature points.

Algorithm 2: Feature Point Matching Selection Between Frames Pseudocode.

Input: p_c : Feature point in current frame
P_{knn} : Feature point in the K nearest neighbor point in the previous frame
Result: $P_{matched}$: K nearest neighbor feature Point selected

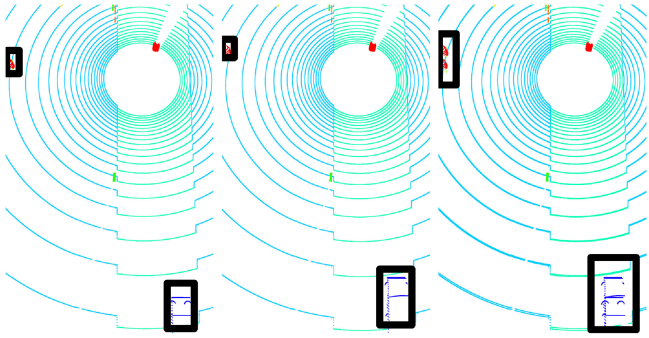
```
Main Loop:
for k = 1 : |P_knn| do
    if p_k.class == p_c.class then
        P_matched ⟵ p_k ;
    end if
end for
return P_matched
```

The Algorithm 1 is based on the semantic information of the point cloud, and the rule of whether the object moves. And the algorithm works after the feature point extraction module performs the preliminary feature point extraction due to the number of the feature point has been reduced by the preliminary selection based on curvature.

3) Feature Point Matching Selection Between Frames: In the lidar odometry and mapping algorithm, the feature points between frames are corresponded by K-Nearest Neighbour method based on the Euclidean distance.However, when the density of the point cloud in the corresponding area between the two frames is different, the nearest neighbor of the European distance K will not be able to accurately match the feature points of the adjacent frame point cloud, especially in the area where the point cloud density is sparse. In the case of mismatching of feature points between frames, an error is generated. In this regard, according to the principle that the semantic information of the same object is unchanged between adjacent point cloud frames, this section expands the K nearest neighbor based on Euclidean distance to Semantic K-Nearest Neighbour. The comparison of semantic labels are added to the judgment process of the distance, as shown in the following Algorithm 2.

4) Motion Trajectory Removal: The point cloud map is formed by merging multi frames point cloud into a map coordinate system by rotating a translation matrix. Therefore, during the construction of the point cloud map, if some objects move relative to others in the scene, different observation values will be left in multiple point cloud frames. When merging the point cloud frames, the moving object will leave a motion trajectory in the point cloud map, as shown in the Fig. 4.

Fig. 4(a), Fig. (b) are two continuous frames point clouds, where the rectangles are the observed values of car and pedestrian in moving. 4(c) is the registered point cloud by the lidar odometry. The registered point cloud scene, which contains multiple observations of moving objects, forms a point cloud trajectory of the moving object. The point cloud trajectory will affect the localization algorithm and reduce the localization accuracy of the autonomous robot. In this regard, this section remove moving objects and their motion trajectories in the point cloud through selecting the point cloud map frames through semantic information, which is shown in the Algorithm 3.



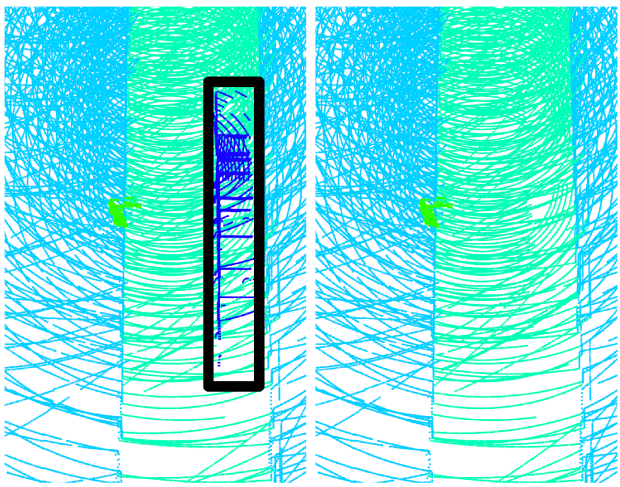
(a) Frame t_n (b) Frame t_{n+1} (c) Registration of frame t_n and t_{n+1}

Fig. 4. Bird view of trajectory of moving objects.

Algorithm 3: Motion Trajectory Removal Pseudocode.

Input: P : Point cloud map
L_{filer} : Label list needed to filter
Result: $P_{filtered}$: Point cloud after trajectory removal

```
Main Loop:
for k = 1 : |P| do
    if p_k.class within L_filer then
        P_filtered ⟵ p_k ;
    end if
end for
return P_filtered
```



(a) Original point clouds (b) The point clouds removing trajectory

Fig. 5. Bird view of the point clouds removing trajectory.

This section takes the autonomous vehicles as an example to remove the motion trajectory in the point cloud map. The common moving objects in the map are pedestrians and cars, so pedestrians and cars are the point cloud class to be filtered. And then the the motion trajectory will be removed based on the filtering result in point cloud map.

Fig. 5 shows the motion trajectory removal result. Fig. 5(a) is the original point cloud with trajectory of a moving car. 5(b) is the result after removing the motion trajectory. As a result of the removal of the motion trajectory, the driving trajectory in the rectangle has been accurately removed. Then Algorithm 3 can provide a static semantic point cloud map without accurate object motion trajectory.

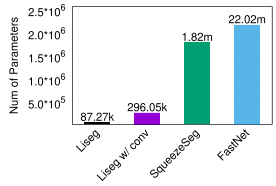


Fig. 6. Comparison of parameter number between models.

IV. Experiment and Result Analysis

In the experiment, raw data of various scenes were collected to test the performance of semantic segmentation module and mapping module. Semantic segmentation module include evaluations of runtime and accuracy. In mapping module we used two common indexes, which were translation error (OdomErrRate) and rotation error (RotErrRate), to evaluate the accuracy of our algorithm.

A. Setup of Semantic Segmentation Network

Point Cloud Data: The experiment is performed with Velodyne Lidar with the frequency at 10Hz and measurement precision at 2cm. We mainly annotated three kinds of common objects, which are road, pedestrian and vehicle. 531 frames and 4436 objects, including 3247 instances of vehicles and 1189 instances of pedestrians, containing 16.07 million points in tatal, were annotated in point-wise level with the help of annotation tools. The point cloud data included sections of busy roads in Beijing representing high dense traffic and less busy road sections in school. In our datasets, the point clouds of vehicles, pedestrians and road accounted for 7.6%, 0.32% and 52.15%, respectively.

Parameters of Network: All the models were implemented based on Tensorflow Estimator API[2] on Nvidia GTX1070 and Nvidia TX2, and the corresponding input and output layers were modified according to semantic annotation of point cloud datasets proposed in this paper. Model's learning rate, batch size, epoch, and fold or cross validation are 0.001, 8, 8, 5 correspondingly.

B. Evaluation of Semantic Segmentation Network

The paper qualified the performance of the point cloud segmentation module, which was a lightweight convolutional neural network, in two aspects: model runtime and model accuracy. Meanwhile, our model was compared with other networks based on Lidar point cloud, including FastNet [23], SqueezeSeg [25] and Liseg w/conv [14] based on common two-dimensional convolution.

[2]https://www.tensorflow.org/programmers_guide/estimators

Model Name	(MB)
Liseg	**0.96**
Liseg w/ conv	2.0
SqueezeSeg	8.5
FastNet	88.5

Fig. 7. File size of model.

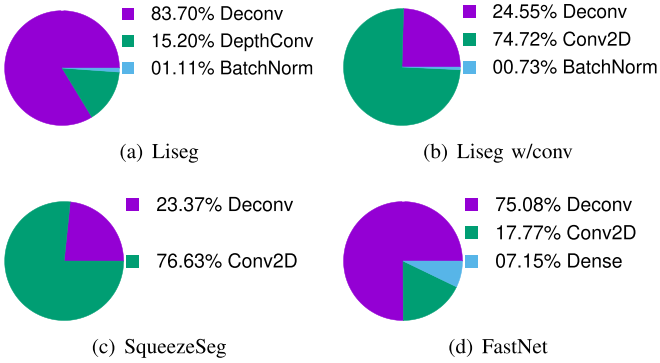


Fig. 8. Parameter distribution of each models.

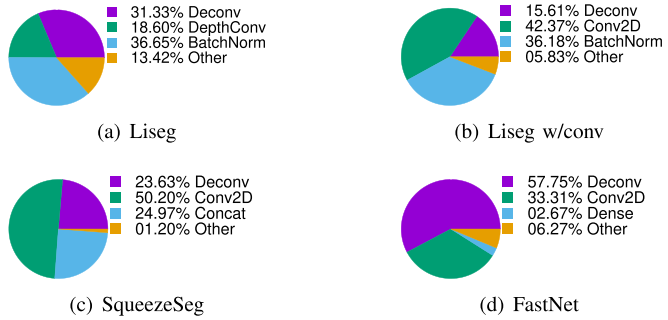


Fig. 9. Distribution of memory usage of each model.

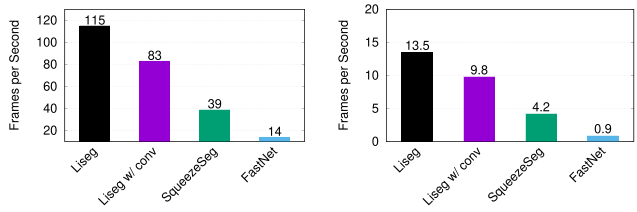


Fig. 10. Inference Frame Rate of Difference Models.

1) Evaluation on Runtime: The evaluation of runtime include two parts, model size and runtime. Model size was essential because our model focused on embedded system of self-driving car, whose computing resources are limited. The model size evaluation and comparison completed three aspects: the number of model parameters, the size of exported files of the model, and memory used in the model. The runtime performances were evaluated in two aspects: inference time on single frame of point cloud and runtime of each layer.

The Number of Model Parameters: To evaluate the number of mode parameters, we used tf.profiler to measure the profile in each layer of model. The result was shown below.

As shown in the table, the parameter number of Liseg w/conv, SqueezeSeg and FastNet are 339%, 2085%, and 25232% of Liseg's, respectively. Through the comparison, the lightweight model Liseg proposed in the paper required much less parameters than other models. Therefore, Liseg was less likely to encounter over-fitting than other models. Moreover, we analysed the distribution of parameters of various operations in each model The result was shown in Fig. 8

The Size of Exported file: The experiment recorded weights and network structure in Protocol Buffer,[3] which was a text format. The comparison of the exported files' size between the models was shown in Fig. 7. As shown in the table, the lightweight model Liseg evidently outperforms other models in the file size due to less parameters.

GPU Memory Usage of the Model: We evaluated the GPU memory usage of each model with the help of the tf.profiler by recording the memory requested in the process of inference. The result was shown in Fig. 9. As the figure shown, Liseg-w/conv's memory usage was nearly twice that of Liseg, and memory usage in SqueezeSeg and FastNet model were above 3 times and 6 times that of Liseg. Therefore, the light-weight model Liseg requested less GPU memory, and it preceded other models in computing resources. The model Lidseg was a more suitable model for self-driving system.

Inference Time: The inference time on a single frame was relative small. For more intuitional comparison of runtime between models, the performances were demonstrated in average inference frames per second. The results were shown in Fig. 10. As the figures shown, on both platforms, our models obviously performed better in runtime than other models. On the platform of Nvidia GTX 1070, our model could averagely process 115 frames of point clouds per second with TDP 150w.in inferences, which was 138% that of Liseg w/conv, 295% that of SqueezeSeg and 821% that of FastNet. On the platform of Nvidia TX2, 13.5 frames of point clouds averagely were processed by our model per seconds with TDP 15w, and the rate equaled to 138% of Liseg w/conv, 321% of SqueezeSeg and 1500% of FastNet. The results revealed that our models had high runtime performance, and could be applied to real-time inference.

Runtime Distribution: We measured the runtime of each layer of our model, the distribution of which was shown in the following Gantt chart (11). From the chart, depth separable convolutional layer and convolutional layer of Liseg had the shorter runtime due to the use of depth-oriented separable convolution and less number of convolution kernels. The main computation time was concentrated in two deconvolution layers, which corresponded to the highest ratio of deconvolution floating-point operations in Liseg. Namely, the bottleneck that limits Liseg to further improve efficiency attributed to large number of floating-point operations in the deconvolution layer. In the future, the inference efficiency of the model could be improved by further optimizing the deconvolution layer.

[3]https://developers.google.com/protocol-buffers/

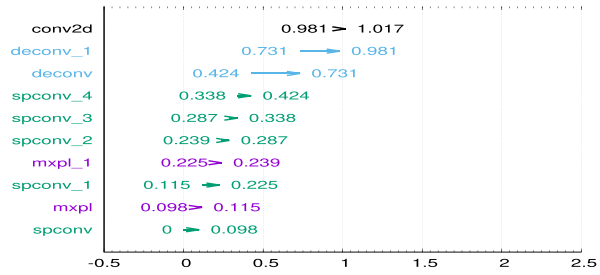


Fig. 11. Distribution of runtime in each layers (in ms). Conv2d is common two-dimensional convolution, deconv denotes decolvolution layers, spconv denotes separable convoluion layer, and mxpl denotes maximum pooling layer.

TABLE II
SEMANTIC SEGMENTATION ACCURACY OF EACH MODEL

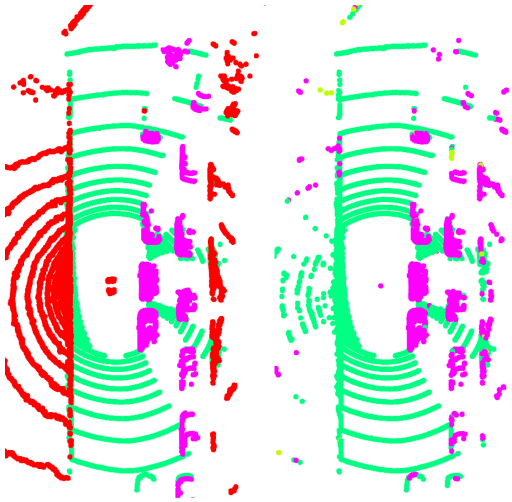
Model	mIoU	mPA	Model	mIoU	mPA
Liseg	**0.76**	**0.81**	Liseg w/ conv	0.70	0.72
SqueezeSeg	0.70	0.73	Liseg w/o mp	0.70	0.72
FastNet	0.68	0.69	Liseg w/ fmp	0.65	0.68
Liseg w/ swce	0.70	0.72	Liseg w/ sr	0.68	0.71
Liseg w/o wce	0.65	0.70			

2) Evaluation on Model Accuracy: The accuracy of our semantic segmentation model was evaluated on two common indexes: Mean Intersection over Union (mIoU) and Mean Pixel Accuracy (mPA). The formulas to calculate these index was shown in formula 8. The $k+1$ denoted the number of classes, and p_{ij} indicated the number of the points (p) in class i but predicted to class j.

$$mIoU = \frac{1}{k+1}\sum_{i=0}^{k}\frac{p_{ii}}{\sum_{j=0}^{k}p_{ij}+\sum_{j=0}^{k}p_{ji}-p_{ii}}$$
$$mPA = \frac{1}{k+1}\sum_{i=0}^{k}\frac{p_{ii}}{\sum_{j=0}^{k}p_{ij}} \quad (8)$$

We tested various kinds of Lidar-based semantic segmentation models, including SqueezeSeg, FastNet and some Liseg models with different implementations. The results were shown in Table II. From the table we can see that the mIoU and mPA of the Liseg are 0.76 and 0.81, respectively, which were both higher than that of SqueezeSeg and FastNet. Meanwhile, the mIoU of the model Liseg w/swce based on the loss function with single-scale weighted cross entropy was 0.7, and the mIoU of the model Liseg w/o wce based on loss function with common cross entropy was 0.66 due to imbalance between classes. The mIoU of Liseg w/ conv using common two-dimensional convolution and Liseg w/omp using large stride in two-dimensional convolution instead of reducing feature in pooling layer were both 0.7. The mIoU of the model Liseg w/sr using feature map pooling in two dimension dropped to 0.65 because of loss of many features. For lack of global information, the mIoU of Liseg w/sr using small receptive field was 0.68. In general, the mIoU and mPA of Liseg using loss function of weighting cross entropy in multiple layers both outperformed the other models.

The intuitional result of semantic segmentation was shown in Fig. 12. Comparing the segmented point clouds with the labeled ones, the lightweight model Liseg could successfully and correctly classify the different objects.



(a) Bird view of reference (ground truth) (b) Bird view of segmented point cloud using Liseg

Fig. 12. The result of point cloud segmentation. The red point clouds denoted the areas of Not Care, green point clouds represented the road the purple point clouds were vehicles.

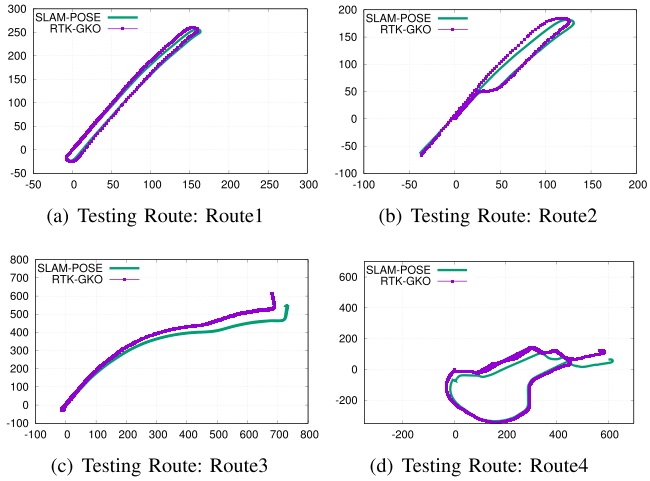
C. Setup of Mapping Module

Datasets: The datasets mainly contains point clouds and corresponding GPS (Global Postion System) information. The data of point cloud was obtained from Velodyne Lidar with the frequency at 10 Hz and measurement precision at 2 cm. The GPS information was acquired by differential correction using RTK (Real Time Kinematic), and it combined the information of OxTS Inetial + 2 Inertial Measurement Unit (IMU) through Kalman filter. The frequency and measurement precision of GPS are 100 Hz and 1cm, respectively. These two sensors were synchronized by satellite time and their data were saved in the form of stream.

Parameters Initialization: For any GPS and Lidar data stream in testing routes, $\boldsymbol{PCF}$ denoted the sequence of point cloud stream, and $\boldsymbol{GK} = [x_g, y_g, h_g]$ contained GPS coordinates in the data stream and coordinate sequences pointing to ground after Gaussian projection. $\boldsymbol{GK'}$ was the subset of $\boldsymbol{GK}$. For any $\boldsymbol{PCF_i}$, $\boldsymbol{GK'_i}$ was the ground coordinate synchronous to point cloud frame $\boldsymbol{PCF_i}$. $\boldsymbol{GKO_i} = [x_o, y_o, h_g]$ was the ground coordinate from translation of $\boldsymbol{GK'_i}$ with $\boldsymbol{GK'_0}$ as origin. $\boldsymbol{pose_i} = [x_v, y_v, h_v]$ denoted the pose of vehicle after the processing of point cloud frame $\boldsymbol{PCF_i}$. The initial conditions were: $\boldsymbol{GKO_0} = [0, 0, h_g]\boldsymbol{pose_0} = [0, 0, h_v]$ and $h_v = h_g$.

D. Evaluation of Mapping Module

In this section, we would demonstrate the results of our odometry algorithm. We used two common indexes, which were translation error and rotation error, to evaluate the accuracy of our algorithm. The translation in this module was compared with the translation after Gaussian projection of latitude and longitude, namely ground truth.



(a) Testing Route: Route1 (b) Testing Route: Route2

(c) Testing Route: Route3 (d) Testing Route: Route4

Fig. 13. The trajectory of testing routes and corresponding reference. SLAM-POSE were trajectory using our mapping module, and RTK-GKO were corresponding reference.

For all testing routes, our evaluation computed translational and rotational error with following formula. The $\mathrm{Odom_{SLAM}}$ denoted translation of our odometry, and $\mathrm{Odom_{RTK}}$ was the reference. Errors (OdomErrRate and $\mathrm{RotErrRate}$ are measured in percent (for translation) and in radians per meter (for rotation).

$$\begin{aligned} \mathrm{Odom_{SLAM}} &= \sum_{i=0} dist(\boldsymbol{pose_{i+1}}, \boldsymbol{pose_i}) \\ \mathrm{Odom_{RTK}} &= \sum_{i=0} dist(\boldsymbol{GKO_{i+1}}, \boldsymbol{GKO_i}) \\ \mathrm{OdomErrRate} &= \frac{|\mathrm{Odom_{SLAM}} - \mathrm{Odom_{RTK}}|}{\mathrm{Odom_{RTK}}} \end{aligned} \tag{9}$$

Rotation error $\mathrm{RotErrRate}$:

$$\mathrm{RotErrRate} = \frac{|\sum_{i=0}(\boldsymbol{pose_i}.h_v - \boldsymbol{GKO_i}.h_g)|}{\mathrm{Odom_{RTK}}} \tag{10}$$

Mapping Result: We tested our mapping module with four routes represented as route1, route2, route3 and route4, the result of which were shown in Fig. 13.

Comparing 13(a) and (b), the trajectory of SLAM-POSE nearly coincided with reference trajectory of RTK-GKO. Meanwhile, the result trajectory of our mapping module in other three routes were relatively same as the reference. The quantified results before and after feature selections using semantic information were shown in Table III.

From the table, our Lidar based odometry algorithm used in mapping module obtained high precision, especially in route1 and route2. Because our mapping module had not integrated closure detection and the length of route3 and route4 reached to 2 kilometer with mass rotations, the rotation errors accumulated up to 0.3 (rad/m) and 0.41 (rad/m) respectively.

The semantic mapping results were shown in Fig. 14. As shown in Fig. 14, different objects around the vehicle were accurately segmented, and our system could efficiently and accurately build the semantic point cloud map.

TABLE III
ACCURACY OF MAPPING MODULE

Testing Route	reference tranlation (m)	mapping translation (m)	translational error (ratio)	rotational error (radian per meter)
route1 (before)	748.75	743.70	0.68%	0.07
route1 (after)	748.75	745.46	**0.44%**	**0.06**
route2 (before)	536.05	530.94	**0.95%**	0.03
route2 (after)	536.05	530.37	1.06%	0.03
route3 (before)	2132.60	2123.22	0.44%	0.40
route3 (after)	2132.60	2126.82	**0.27%**	**0.35**
route4 (before)	2269.02	2238.39	1.35%	0.41
route4 (after)	2269.02	2243.89	**1.11%**	**0.40**

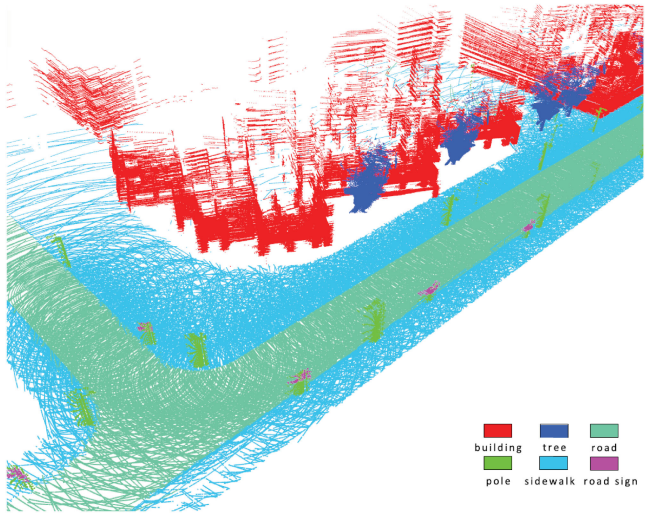


Fig. 14. Semantic mapping results.

V. CONCLUSION

We propose a lightweight semantic segmentation network to assist in localization and mapping in this paper. The method uses the lidar point clouds generated by the simulator and annotated manually in real world as the original input. Then, the semantic segmentation network LiSeg segment the point clouds to obtain the semantic information. Finally, the semantic information is used in LOAM-SLAM to assist the localization and mapping.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.
[2] R. Zlot and M. Bosse, "Efficient large-scale 3D mobile mapping and surface reconstruction of an underground mine," in *Field and Service Robotics*. Berlin, Germany: Springer, 2014, pp. 479–493.
[3] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2174–2181.
[4] J. Deschaud, "IMLS-SLAM: Scan-to-model matching based on 3D data," *CoRR*, vol. abs/1802.08633, 2018.
[5] A. Sangiovanni-Vincentelli and M. D. Natale, "Embedded system design for automotive applications," *Computer*, vol. 40, no. 10, pp. 42–51, 2007.
[6] L. Chen, M. Cui, F. Zhang, B. Hu, and K. Huang, "High-speed scene flow on embedded commercial off-the-shelf systems," *IEEE Trans. Ind. Inform.*, vol. 15, no. 4, pp. 1843–1852, Apr. 2019.
[7] K. Huang, B. Hu, L. Chen, A. Knoll, and Z. Wang, "AdasonCotswith openCL: A case study with lane detection," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 559–565, Apr. 2018.

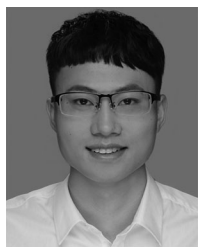
[8] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," *Auton. Robots*, vol. 41, no. 2, pp. 401–416, 2017.
[9] D. Maturana and S. Scherer, "Voxnet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 922–928.
[10] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1513–1518.
[11] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "Semantic3d.net: A new large-scale point cloud classification benchmark," in *Proc. ISPRS Ann. Photogrammetry, Remote Sens. Spatial Inf. Sci.*, vol. IV-1-W1, 2017, pp. 91–98.
[12] A. Dewan, G. L. Oliveira, and W. Burgard, "Deep semantic classification for 3D lidar data," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 3544–3549.
[13] M. Velas, M. Spanel, M. Hradis, and A. Herout, "CNN for very fast ground segmentation in velodyne LiDAR data," in *Proc. IEEE Int. Conf. Auton. Robot Syst. Competitions*, 2018, pp. 97–103.
[14] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSeg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3D lidar point cloud," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1887–1893.
[15] A. Borkowski, B. Siemiatkowska, and J. Szklarski, *Towards Semantic Navigation in Mobile Robotics*. Berlin, Germany: Springer, 2010, pp. 719–748.
[16] R. Horaud, M. E. Hansard, G. D. Evangelidis, and C. Ménier, "An overview of depth cameras and range scanners based on time-of-flight technologies," *Mach. Vis. Appl.*, vol. 27, no. 7, pp. 1005–1020, 2016.
[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.
[18] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," *Auton. Robots*, vol. 41, no. 2, pp. 401–416, 2017.
[19] I. Cvišić, J. Ćesić, I. Marković, and I. Petrović, "Soft-slam: Computationally efficient stereo visual slam for autonomous UAVs," *J. Field Robot.*, vol. 35, no. 4, pp. 578–595, 2018.
[20] M. Buczko and V. Willert, "Flow-decoupled normalized reprojection error for visual odometry," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2016, pp. 1161–1167.
[21] J. Zhu, "Image gradient-based joint direct visual odometry for stereo camera," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 4558–4564.
[22] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
[23] A. Dewan, G. L. Oliveira, and W. Burgard, "Deep semantic classification for 3D lidar data," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3544–3549.
[24] G. L. Oliveira, W. Burgard, and T. Brox, "Efficient deep models for monocular road segmentation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 4885–4891.
[25] G. Nanfack, A. Elhassouny, and R. O. H. Thami, "Squeeze-segnet: A new fast deep convolutional neural network for semantic segmentation," *CoRR*, vol. abs/1711.05491, 2017.
[26] J. Xiao, A. Owens, and A. Torralba, "Sun3D: A database of big spaces reconstructed using SFM and object labels," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 1625–1632.
[27] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 3050–3057.
[28] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up slam: Semantic monocular plane slam for low-texture environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1222–1229.
[29] B. Hua, Q. Pham, D. T. Nguyen, M. Tran, L. Yu, and S. Yeung, "Scenenn: A scene meshes dataset with annotations," in *Proc. 4th Int. Conf. 3D Vis.*, 2016, pp. 92–101.
[30] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic slam," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1722–1729.
[31] S. Yang, Y. Huang, and S. Scherer, "Semantic 3D occupancy mapping through efficient high order CRFs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 590–597.
[32] Y. Xiang and D. Fox, "DA-RNN: Semantic mapping with data associated recurrent neural networks," arXiv:1703.03098, 2017.
[33] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3D semantic mapping with convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4628–4635.
[34] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object classification using CNN-based fusion of vision and lidar in autonomous vehicle environment," *IEEE Trans. Ind. Inform.*, vol. 14, no. 9, pp. 4224–4231, Sep. 2018.
[35] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1800–1807.

Zhihao Zhao received the B.Sc. degree from Sun Yat-sen University, Guangzhou, China, in 2018, where he is currently working toward the M.Sc. degree.

His research interests include computer vision, especially in 3-D point cloud detection and semantic segmentation, autonomous driving based on deep learning.

Wenquan Zhang received the B. Sc. and M.Sc. degrees from Sun Yat-sen University, Guangzhou, China, in 2016 and 2018, respectively.

His research interests include computer vision, especially in 3-D point cloud processing and semantic segmentation, autonomous driving based on deep learning.



Jianfeng Gu received the B.Sc. degree from Sun Yat-sen University, Guangzhou, China, in 2018, where he is currently working toward the M.Sc. degree.

His research interests include computer vision, especially in 3-D point cloud processing and reconstruction, deep learning, and sensor fusion in autonomous driving areas.

Junjie Yang received the B.Sc. degree from Sun Yat-sen University, Guangzhou, China, in 2018, where he is currently working toward the M.Sc. degree.

His research interests include safety-critical embedded systems and vehicle-mounted LiDAR point-cloud processing integrated with machine learning.

Kai Huang received the B.Sc. degree from Fudan University, Shanghai, China, in 1999, M.Sc. degree from University of Leiden, Leiden, the Netherlands, in 2005, and the Ph.D. degree from ETH Zurich, Zurich, Switzerland, in 2010.

He was a Professor with the Sun Yat-Sen University, Guangzhou, China, in 2015. He was appointed as the Director with the Institute of Unmanned Systems of School of Data and Computer Science, in 2016. He was a Senior Researcher with the Computer Science Department, Technical University of Munich, Munich, Germany from 2012 to 2015 and a Research Group Leader with fortiss GmbH, Munich, Germany, in 2011. His research interests include techniques for the analysis, design, and optimization of embedded/CPS systems, particularly in the automotive and robotic domains.