Sensor-independent Pedestrian Detection for Personal Mobility Vehicles in Walking Space Using Dataset Generated by Simulation*

Takahiro Shimizu†, Kenji Koide‡, Shunji Oishi‡, Masashi Yokozuka‡, Atsuhiko Banno‡ and Motoki Shino†
†Department of Human and Engineered Environmental Studies, Graduate School of Frontier Sciences, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba, Japan.
Email: shimizu.takahiro@atl.k.u-tokyo.ac.jp, motoki@k.u-tokyo.ac.jp
‡Human-Centered Mobility Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Central 2, 1-1-1, Umezono, Tsukuba, Ibaraki 305-8568, Japan.
Email:{k.koide, shuji.oishi, yokotsuka-masashi, atsuhiko.banno}@aist.go.jp

***Abstract*—Autonomous driving of a personal mobility vehicle such as a wheelchair in a walking space is crucial in the future as a means of transportation for the elderly and the physically handicapped. To realize this, accurate pedestrian detection is indispensable. As existing 3D object detection methods are trained with a roadway dataset, they are widely used for object detection in roadways. These methods have two major drawbacks as regards the detection of objects in walking spaces. The first is that they are largely depend on the different LIDAR models. To eliminate this issue, we propose a 3D object detection method, CosPointPillars, that does not take the reflection intensities of the LIDAR point cloud, which causes a sensor model dependency, as input. The second drawback is that networks trained with a roadway dataset cannot sufficiently detect pedestrians (who are major traffic participants in walking spaces) located within a short distance; this is because the roadway dataset hardly includes nearby pedestrians. To solve this issue, we generated a new walking space dataset called SimDataset, which includes nearby pedestrians as a training dataset in the simulations. An experiment on a real walking space showed that SimDataset is suitable for use in pedestrian detection.**

I. INTRODUCTION

Autonomous driving of a personal mobility vehicle such as a wheelchair in walking spaces is necessary as a means of transportation for the elderly and the physically handicapped. To realize this, accurate pedestrian detection is indispensable for collision avoidance. In recent years, many LIDAR-based 3D object detection methods using convolutional neural networks (CNN) have been proposed. Being trained with a roadway dataset, these methods are widely used for object detection on roadways.

Existing methods trained with a roadway dataset have two major issues when applied to wheelchairs. The first issue is that these networks are largely affected by the difference of LIDAR models. This is because the input of most of the existing methods includes the reflection intensity of a LIDAR point cloud, whose characteristics largely depend on the sensor model [19][20]. LIDARs equipped with wheelchairs are different from those equipped with cars. LIDARs on wheelchairs are smaller and have fewer scan lines compared with those on cars, owing to price and size restrictions. For example, it is difficult to mount Velodyne HDL-64E on a wheelchair, which is a LIDAR widely used for cars, and only a small LIDAR can be mounted as shown in Fig. 1. Therefore, it is not appropriate to apply the datasets of the LIDARs mounted on cars to wheelchairs.

Fig. 1. Example of a personal mobility vehicle in walking space. The wheelchair is equipped with a relatively small LIDAR.

The second issue is that the surrounding environment of cars and wheelchairs is different. In particular, there are few pedestrians near cars on roadways, while there are many pedestrians near wheelchairs on walking spaces. Therefore, roadway datasets are not optimal for object detection in walking spaces. There are, however, not enough walking space datasets containing LIDAR point clouds.

In this study, we realized accurate pedestrian detection in walking spaces by using a sensor-independent network trained with a walking space dataset. To eliminate the sensor model dependence issue, we created a 3D object detection network, whose input does not include reflection intensities of LIDAR points. This network includes an architecture called cosine estimation network (CEN) to retain the detection accuracy.

*This work was partially supported by a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and a JSPS KAKENHI(Grant Number 18K18072).

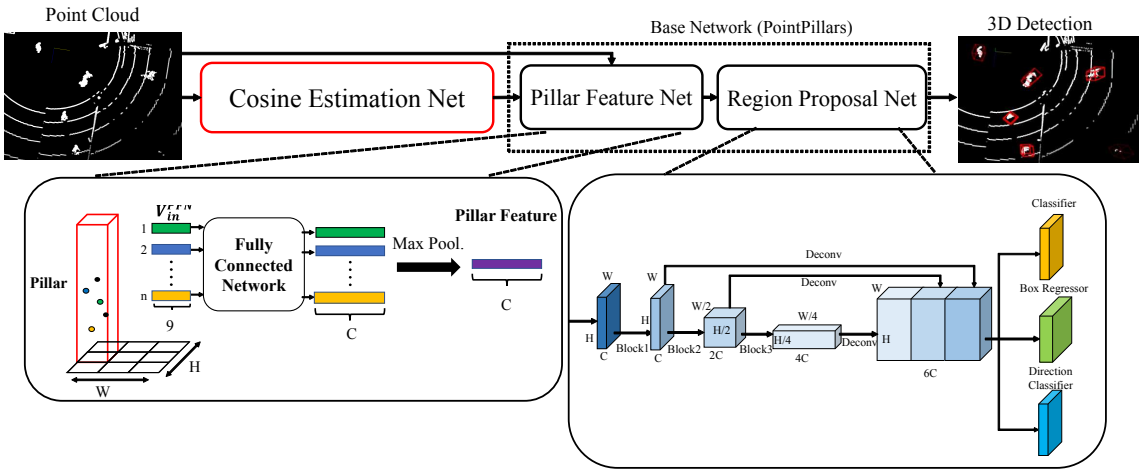

Fig. 2. Schematic of architecture of CosPointPillars

Furthermore, because there are not enough walking space datasets, we generated a new walking space dataset that includes nearby pedestrians, using a simulation on a game engine.

The contribution of this study is two-fold. First, we propose a 3D object detection network independent of reflection intensity. Second, we propose a walking space dataset generated by simulation. Through an experiment, we showed that the proposed network trained with our dataset achieved high pedestrian detection accuracy in a real walking space.

The rest of this study is organized as follows. Section II reviews related studies on 3D object detection methods and datasets. Section III describes the proposed pedestrian detection network with a CEN. Section IV explains the details of our walking space dataset. Section V shows the results of two experiments: one shows that the CEN helps to retain the detection accuracy while removing the sensor dependency, and the other demonstrates that the proposed network is applicable to real walking spaces.

II. RELATED WORK

A. 3D object detection

Many CNN-based 3D object detection methods have been proposed. Some of them use LIDAR point clouds projected on planar maps. MV3D [1] is a pioneering method that uses a bird's eye view (BEV) mapping approach. The problem with this method is that it is difficult to recognize relatively small objects such as pedestrians. There are other methods that utilize BEV maps [2][3][4]. AVOD [2] shows that applying a feature pyramid network improves the detection accuracy for small objects.

Other methods generate regions of interest and directly detect objects based on the points in the region [7][8][15]. Frustum PointNets [7] and Frustum ConvNet [8] generate frustum-shaped regions of interest based on 2D object detection on RGB images. Meanwhile, PointRCNN [15] generates regions of interest using the whole point cloud.

There are methods that use a voxelized representation of LIDAR point clouds [9][10][11]. VoxelNet divides the detection space into voxels, and generates the features of each voxel with a network inspired by PointNets [5][6]. PointPillars [11] divides the space into columns (also known as pillars) that extend from 2D grids on the horizontal plane in the vertical direction. These methods use only point cloud as the input and do not require RGB images.

Amongst existing 3D object detection methods, we focused on PointPillars, which is particularly excellent in terms of pedestrian detection accuracy and speed. The original PointPillars takes reflection intensities of point clouds as input. This leads to the limitation of versatility, because the characteristics of reflection intensity measurements largely depend on the sensor model.

B. Dataset

Roadway datasets are utilized for training and evaluating 3D object detection networks. These datasets provide sensor data with object annotations, e.g. pedestrians, cars, cyclists. KITTI dataset [12] is the pioneering multimodal roadway dataset that provides LIDAR point clouds, front-facing images, and GNSS/IMU data. In KITTI, only frontal objects are annotated. H3D [13] and NuScenes [14] are datasets providing a large number of diverse scenes and objects. Furthermore, in these datasets, objects in all directions are annotated. However, these roadway datasets are not optimal for object detection in walking spaces because the roadway and walking space environments are largely different. In particular, roadway datasets contain few pedestrians (who are major traffic participants in walking spaces) within close proximity.

To train a 3D object detection network, it is also possible to use datasets generated by simulation. There are two major advantages of generating a dataset by simulation. First, a large amount of perfectly annotated data can be obtained within a short period of time. Second, a simulated environment can be customized freely. AirSim [16] and CARLA [17] are

popular simulators that can generate LIDAR point clouds. They can simulate various environments and objects; however, the objects in the point cloud have quite simplified shapes because they are generated from simple collision models. For realistic LIDAR point cloud simulation, we generated point clouds by performing raycasting on depth images. In this manner, we can obtain point clouds with the same geometric shapes as rendered on RGB images.

III. METHODOLOGY

Most of the existing 3D object detection methods take reflection intensities as input. These methods can deteriorate owing to sensor differences because the characteristics of the reflection intensity depend on the sensor model [19][20].

We propose a sensor-independent network that does not use the reflection intensities as input. We refer to this network as CosPointPillars. Figure 2 shows a schematic of CosPointPillars. Most parts of the architecture follow PointPillars, which is a LIDAR-based end-to-end 3D object detection method showing excellent detection speed and accuracy. PointPillars takes the coordinates and the reflection intensity of each point. CosPointPillars differ from PointPillars in that, the reflection intensity is removed from the input and, that a cosine estimation network (CEN) is newly added.

A. Alternative channel for reflection intensity

In addition to removing the reflection intensity from the input, we alternatively set sensor-independent features of the reflection intensity to retain the detection accuracy. We chose the alternative features from the information that the reflection intensity contained. Assuming that the reflection intensity measurement of the LIDAR is proportional to the spectral illuminance of the reflected light, and that the reflection follows the Lambertian model, the reflection intensity r is modeled as follows:

$$r = \frac{kK_\lambda \cos\theta}{d^2}, \tag{1}$$

where k is a proportional constant, K_λ is the reflectivity of the object at wavelength λ, θ is the incident angle of the LIDAR ray on the surface of the object, and d is the distance from the surface to the photoreceptor of the LIDAR.

The reflectivity K_λ is excluded from the candidates of the alternative features because it largely depends on the wavelength of the LIDAR model. The distance d is also not set as an alternative channel. It is because d is a simple feature that can be easily encoded from the coordinates. Furthermore, some sensor models such as Velodyne's LIDARs provide reflection intensities normalized to distance, wherein the distance information is not included.

In this study, we selected $\cos\theta$ as an alternative channel of the reflection intensity. The value of $\cos\theta$ is considered to be a parameter reflecting local characteristics of the reflection intensity. Though real LIDAR reflection intensity does not completely follow ideal Lambertian reflection, it tends to decrease with increasing incident angle θ [19][20]. Furthermore, it is considered to be difficult for the network

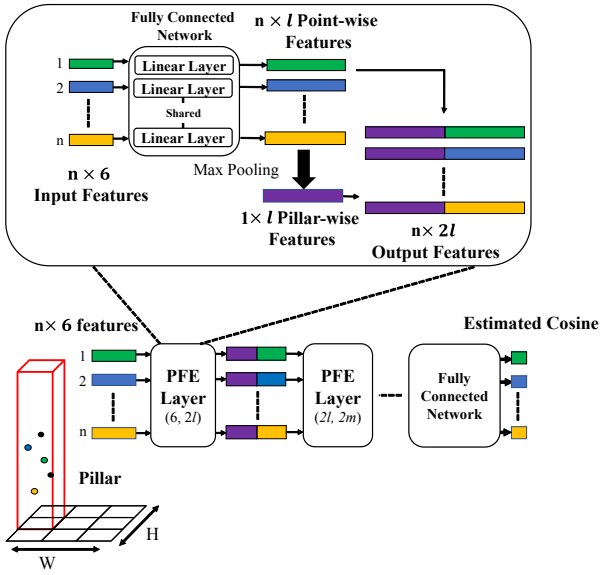

Fig. 3. Schematic of architecture of CEN

to acquire this information naturally, because $\cos\theta$ contains information of the positional relationship with neighboring points. Therefore, even though $\cos\theta$ is a feature that can be encoded from coordinates of points, it is meaningful to estimate $\cos\theta$ explicitly.

B. Cosine estimation network (CEN)

By introducing the CEN, we aim to compensate for the missing geometrical information contained in the reflection intensity. A naive way to make existing networks independent of the reflection intensity is to simply remove it from the input channels. However, this causes a lack of information, which can result in degraded detection accuracy. To retain the detection accuracy, the CEN estimates $\cos\theta$ as an alternative feature of the reflection intensity. The normal of the point, by which $\cos\theta$ is defined, is often calculated from its neighboring points; however, it is difficult to calculate accurate values of the normal from sparse point clouds. Thus, we used a neural-network-based method to robustly estimate $\cos\theta$ from sparse points.

Similar to PointPillars, the detection space is divided into pillars that extend vertically from the grid dividing the XY plane into W H.

First, the points in each pillar are fed into the CEN, which estimates $\cos\theta$ for each point. The structure of the CEN is inspired by VoxleNet's feature learning network [9], consisting of multiple voxel feature encoding layers (VFE layer) in a series. Figure 3 shows a schematic of the CEN for one pillar. The input for the CEN is $V_{in} = \{p_i = [x_i, y_i, z_i, x_i - m_x, y_i - m_y, z_i - m_z]^T \in \mathbb{R}^6\}_{1...n}$, where m_x, m_y, and m_z are the x, y, and z values, respectively, of the average coordinates of the points in the pillar. The value n is the maximum number of features input to one pillar. If the number of points in a pillar is larger than n, random sampling is performed. Conversely,

if the number of points is smaller than n, zero padding is performed.

Next, p_i is converted to point-wise pillar features $f_i^{PFE} \in \mathbb{R}^{2l}$ through the first pillar feature encoding (PFE) layer. The PFE layer has a similar structure to the VFE layer of VoxelNet. In the first PFE layer, p_i is converted to $f_i^{point} \in \mathbb{R}^l$ through a fully connected network. This fully connected network consists of a linear layer, batch normalization (BatchNorm) method, and a rectified linear unit (ReLU). Note that, all the linear layers in a fully connected network share their weights as a shared multi-layer perceptron in PointNet [5]. Thereafter, an f_i^{point} is converted to $f_i^{pillar} \in \mathbb{R}^l$ using element-wise max pooling. Finally, a concatenated feature $f_i^{PFE} = [f_i^{point}, f_i^{pillar}]^T$ is the output. In the CEN, multiple PFN layers that have the same architecture as the first one can be chained. In this study, three sequential layers are used. The input and output sizes of each layer are (6, 64), (64, 32), and (32, 8), respectively. The point-wise feature from the last PFE layer is converted to $\cos\theta_i^{est} \in \mathbb{R}$ through a fully connected network.

C. Pillar feature network

Pillar feature net (PFN) has the same structure as that of PointPillars, and it is constructed with a simplified PointNet. The input of this network is $V_{in}^{PFN} = [p_i^T, x_i - c_x, y_i - c_y, \cos\theta_i^{est}]^T \in \mathbb{R}^9\}$, where c_x and c_y are the X and Y coordinates of the center of each pillar, respectively. PFN of CosPointPillars differs from that of PointPillars in that $\cos\theta_i^{est}$ is input instead of the reflection intensity. Thereafter, V_{in}^{PFN} is converted into pillar feature $v_{out}^{PFN} \in \mathbb{R}^C$ through PFN. The value of C can be set arbitrarily, and we set C = 64 in this study. Finally, a feature whose size is (W, H, C) is generated by combining the pillar features calculated in each pillar.

D. Region proposal network

We adopted an SSD-like [18] network proposed in SECOND [10] as a region proposal network (RPN). The RPN consists of three top-down blocks and three deconvolution layers as shown in Fig. 2. A top-down block is a combination of multiple 2D CNN layers. The blocks in CosPointPillars are denoted by block1 to block3. Block1 is a chain of four 2D CNN layers. Block2 is a chain of six 2D CNN layers. Block3 is a chain of six 2D CNN layers. Each of the layers in the blocks is followed by BatchNorm and ReLU. These blocks generate features with size (W, H, C), (W/2, H/2, 2C), and (W/4, H/4, 4C), respectively.

Through the deconvolution layer, features generated from top-down blocks are transformed into features with size (W, H, 2C). Next, BatchNorm and an ReLU are applied to each of these features. By concatenating them, a feature map with size (W, H, 6C) is output.

Finally, the feature map is mapped to three learning targets: a classifier, a box regressor, and a direction classifier. In the box regressor, matching to the ground truth is determined by whether the 2D IoU [22] between the generated bounding box and the ground truth exceeds the threshold.

E. Loss function

The loss function is shown as follows:

$$L = \beta_{det} L_{det} + \beta_{cos} L_{cos}, \tag{3}$$

where L_{det} is a loss function for 3D object detection, and L_{cos} is a loss function for the cosine estimation in the CEN. β_{det} and β_{cos} are constants. In this study, we set $\beta_{det} = 1$ and $\beta_{cos} = 0.5$.

The loss function for the detection of L_{det} is the same as that of Pointpillars and SECOND, and it is defined as follows:

$$L_{det} = \frac{1}{N_{pos}}(\beta_{cls} L_{cls} + \beta_{loc} L_{loc} + \beta_{dir} L_{dir}), \tag{4}$$

where β_{loc}, β_{cls}, and β_{dir} are constants. In this study we set $\beta_{cls} = 1$, $\beta_{loc} = 2$ and $\beta_{dir} = 0.2$. N_{pos} is the number of positive anchors. The function denoted by L_{cls} is the loss function for the classification, and is defined as follows using focal loss [21]:

$$L_{cls} = \sum_{i=1...N_{pos}+N_{neg}} -\kappa(1-p_t)^\gamma \log p_t, \tag{5}$$

where p_t is the certainty of the classification. N_{neg} is the number of negative anchors. κ and γ are constants. We set $\kappa = 0.25$ and $\gamma = 2$. Meanwhile, L_{loc} is a loss function related to the object locations and defined as follows:

$$L_{loc} = \sum_{i=1...N_{pos}} \sum_{b \in x,y,z,w,h,l,h,\Theta} SmoothL1(\Delta b), \tag{6}$$

subject to

$$\begin{gathered}
\Delta x = \frac{x_{gt} - x_a}{d_a}, \Delta y = \frac{y_{gt} - y_a}{d_a}, \Delta z = \frac{z_{gt} - z_a}{h_a}, \\
\Delta w = \log\frac{w_{gt}}{w_a}, \Delta l = \log\frac{l_{gt}}{l_a}, \Delta h = \log\frac{h_{gt}}{h_a}, \\
\Delta\Theta = \sin(\Theta_{gt} - \Theta_a),
\end{gathered} \tag{7}$$

where (x_{gt}, y_{gt}, z_{gt}, w_{gt}, l_{gt}, h_{gt}, Θ_{gt}) and (x_a, y_a, z_a, w_a, l_a, h_a, Θ_a) are the locations, dimensions, and rotation values of the z-axis for bounding boxes of ground truth and positive anchors, respectively. The value d_a is defined by $\sqrt{x_a^2 + y_a^2}$. The function L_{dir} is the orientation loss function proposed in SECOND [10].

The function L_{cos} is a loss function for cosine estimation that we newly add, and it is defined as follows:

$$L_{cos} = \sum_{i=1...N} ||\cos\theta_i - \cos\theta_i^{est}||_2, \tag{8}$$

where N represents the total number of points in all the pillars. The value $\cos\theta_i^{est}$ is the cosine estimated by the CEN at each point.

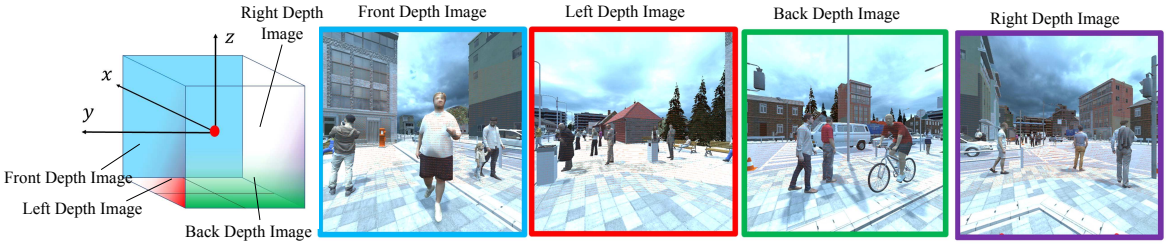

Fig. 4. Schematic of depth images on LIDAR coordinate system.

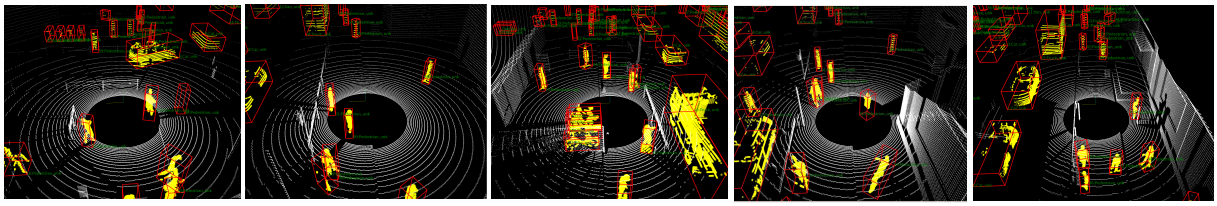

Fig. 5. Examples of generated datasets: In these images, white points represent the generated points and red bounding boxes represent ground truth. Yellow points represent the points in the ground truth bounding boxes.

IV. DATASET

We created a new point cloud dataset in a simulated walking space and referred it to as SimDataset. We created a virtual walking space on Unity, a popular game engine, and simulated point clouds obtained by a LIDAR in the space. There are three major advantages of constructing a dataset by simulation compared with collecting real sensor data. First, a large amount and variety of data with perfect annotation can be generated in a short period of time. Second, the simulated environment can be freely customized depending on the use scenario. Third, accurate cosine can be calculated for the training of CosPointPillars.

SimDataset is generated by performing raycasting on depth images; we can obtain a more realistic point cloud compared with that generated by collision models. Figure 4 shows a schematic of the raycasting-based point cloud generation method.

The definition of the annotation follows that of KITTI. The annotation of each object consists of the class, the location, the dimension, and the rotation angle of the 3D bounding box (Fig. 5). The location is the value of the coordinates (x [m], y [m], z [m]) in the front depth camera coordinate system. The dimension contains the height (h [m]), the width (w [m]), and the length (l [m]) of the 3D bounding box. The rotation angle is the direction of the bounding box about the y-axis in the front depth camera coordinate system.

V. EXPERIMENT

In the first of the two experiments, we compared the pedestrian detection accuracy of CosPointPillars with other methods on downsampled KITTI and NuScenes to show that the cosine estimation helps in pedestrian detection on sparse point clouds. In the second experiment, we compared two CosPointPillars models, which were separately trained with SimDataset and KITTI, to confirm that SimDataset is suitable for pedestrian detection in a walking space.

TABLE I
NUMBER OF PEDESTRIANS PER DISTANCE IN EACH DATASET

dataset	Distance [m]			
	[0, 2.5)	[2.5, 5.0)	[5.0, 7.5)	[7.5, 10.0)
KITTI-train	0	27	112	373
KITTI-test	0	37	224	407
Sim-train	8901	21621	22441	20868
Sim-test	5952	12059	11205	8844

A. Setup of dataset

In these experiments, we used KITTI dataset, NuScenes dataset, and SimDataset for both training and verification. In the SimDataset, we simulated a real LIDAR model equipped with wheelchairs (e.g., Velodyne VLP-16) with 16 scan layers. We generated two sets of data for training and verification: Sim-train and Sim-test. Sim-train and Sim-test contain 12,080 and 10,524 frames, respectively. For these point clouds, pedestrians within 10 m from the virtual sensor are annotated. Assuming that the walking speed of the pedestrian is 4 km/h and the moving speed of the wheelchair is 6 km/h, the time to collide with a pedestrian who is 10 m away is at least 3.6 s. Since this is enough time for collision avoidance, detection of pedestrians who are further away is not required in our case. Sim-train and Sim-test contain 73,831 and 38,060 pedestrian annotations, respectively. Note that we generated these two datasets from different sequences in the simulation to avoid the same pedestrians being included in both datasets.

KITTI dataset provides point cloud data of 7,481 frames for training and 7,518 frames for testing. Since the correct anno-

tations for the testing frames were not available, we only used training frames in the experiments. Following [1][9][10][11], we divided the training frames into 3,712 and 3,769 frames for training and verification such that they did not contain pedestrian data sampled from the same sequence. In this paper, we refer to them as KITTI-train and KITTI-test. Following SimDataset, the point clouds in KITTI were downsampled so that the vertical resolution would become 2° to imitate point clouds of VLP-16. Furthermore, the annotation of objects further than 10 m was removed. To generate the ground truth of $\cos\theta$, we calculated the normal vectors by performing principal components analysis (PCA) of the points within a radius of 0.2 m from each point. Thereafter, the normal was defined by the third component. In the case where the number of surrounding points is insufficient to perform PCA, we define $\cos\theta$ as 0. Note that cosine calculation is performed based on original point clouds without downsampling.

To highlight the difference in the pedestrian distance distributions of KITTI and SimDataset, we summarize the number of pedestrians in different distance ranges, as shown in Table I. SimDataset contains more pedestrians who are nearby compared with KITTI. In particular, while no pedestrian exists within 2.5 m in the KITTI dataset, Sim-train and Sim-test contain 8,901 and 5,952 pedestrians, respectively, located closer than 2.5 m. SimDataset contains a total of 111,891 pedestrians. Note that, while the original KITTI contains a total of 4,487 pedestrians, it contains only 1,180 pedestrians within 10 m.

We also used the NuScenes dataset. NuScenes dataset provides 34,149 frames from 850 different scenes. We following NuScenes' official split, and used 700 scenes and 150 scenes as training and testing data. The point clouds and annotations in NuScenes were downsampled in the same way as in KITTI. The ground truth of $\cos\theta$ was also calculated with the same method as in KITTI.

B. Setup of CosPointPillars

The pillar size was $0.16\times0.16\times5.00$ m^3, and the observation ranges in the X, Y, and Z axes were (0.00, 10.24) m, (-10.24, 10.24) m, and (-2.50, 2.50) m, respectively. The maximum number of points in a pillar was set to 50. The other parameters follow those of the original PointPillars.

In training, we applied the same data augmentation method as that proposed in PointPillars. For KITTI, we applied rotation noise in the range $[-\pi/20, \pi/20]$ to the ground truth bounding boxes and location noise following normal distribution $\mathcal{N}(0, 0.25)$. In addition, global rotation noise in the range $[-\pi/4, \pi/4]$ and uninformed scaling noise in the range [0.95, 1.05] were applied to entire point clouds. Meanwhile, for SimDataset, we only applied global rotation noise in the range $[-\pi/8, \pi/8]$ and uninformed scaling noise in the range [0.95, 1.05] for entire point clouds. Since the SimDataset had enough variation for training, other data augmentation methods were not applied. In these conditions, each of the networks was trained for 200 epochs with a batch size of 2 for KITTI and 6 for SimDataset.

C. Performance of CosPointPillars

First, to show that the CosPointPillars improves the pedestrian detection accuracy in sparse point clouds, we compared the pedestrian detection accuracy of CosPointPillars with existing networks on downsampled KITTI and NuScenes. We evaluated the average precision (AP) in 3D and BEV pedestrian detection. Following the official KITTI evaluation protocol, we set the IoU threshold at 0.5 in the experiment on KITTI. In the experiment on NuScenes, we prepared two types of thresholds of IoU, 0.5, which is the same as that for the KITTI evaluation, and 0.15, which is a more relaxed condition. This is because NuScenes provides more challenging and various testing data compared with KITTI.

Table II lists the AP of 3D and BEV pedestrian detection of CosPointPillars along with that of competing networks. As competing networks, we chose VoxelNet, SECOND, and PointPillars for the evaluation on KITTI and PointPillars for the evaluation on NuScenes. Note that the inputs to these networks were X, Y, and Z coordinates of each point, whereas the inputs of the original ones also included a reflection intensity. In addition, to investigate the effect of removing the reflection intensity from the input, the performance of the original PointPillars on KITTI is also shown in Table II. In the evaluation on KITTI, the AP at each detection difficulty level (Easy, Moderate, and Hard) is calculated. In the evaluation on NuScenes, the AP under each IoU threshold level is calculated.

Regarding BEV pedestrian detection, CosPointPillars outperformed other networks at all difficulty levels on KITTI. On NuScenes, CosPointPillars outperformed PointPillars under both IoU thresholds. The result that CoSPointPillars outperformed PointPillars in BEV detection accuracy on both datasets suggests that the CEN in CosPointPillars contributes to improving the accuracy of BEV pedestrian detection.

As regards 3D pedestrian detection, as TableII shows, CosPointPillars outperforms VoxelNet and SECOND at all the difficulty levels on KITTI. Comparing the CosPointPillars and PointPillars, the AP of the CosPointPillars was higher than that of PointPillars when the detection difficulty level was Easy, and it was lower at both Moderate and Hard difficulty levels. These differences in AP were negligible at all the detection difficulty levels, i.e., less than 0.3%. Thus, the performances of both networks on KITTI are considered to be almost the same. On NuScenes, PointPillars exceeded CosPointPillars in detection accuracy under an IoU threshold of 0.5, whereas CosPointPillars outperformed PointPillars under a threshold of 0.15. These results suggest that the estimation accuracy of the position and size along the Z axis was not improved by the CEN. The most possible cause of the results is that the input 16-layer LIDAR point cloud was sparse in the Z axis. It can be assumed that the CEN could not sufficiently capture the relationships between points in the Z axis because of the sparsity of point cloud in the Z axis.

Comparing PointPillars with and without reflection intensity input, the original PointPillars outperform PointPillars without reflection intensity input in both 3D and BEV pedestrian detec-

TABLE II
AP OF PEDESTRIAN DETECTION ON KITTI AND NUSCENES

Method	Input	KITTI 3D (%)			NuScenes 3D (%)		KITTI BEV (%)			NuScenes BEV (%)		Time
		Easy	Mod.	Hard	IoU 0.5	IoU 0.15	Easy	Mod.	Hard	IoU 0.5	IoU 0.15	(ms)
PointPillars	x, y, z, ref.	**75.17**	**73.16**	**69.58**	—	—	**84.36**	**79.97**	**76.82**	—	—	8
VoxelNet	x, y, z	58.05	54.40	51.74	—	—	66.78	61.97	60.58	—	—	29
SECOND	x, y, z	72.31	67.00	64.36	—	—	79.99	74.97	72.71	—	—	44
PointPillars	x, y, z	74.12	**71.17**	**68.27**	**21.87**	55.10	80.22	75.92	74.48	30.23	55.29	8
CosPointPillars	x, y, z	**74.42**	71.10	68.02	17.85	**56.34**	**82.35**	**77.29**	**75.94**	**32.83**	**56.94**	10

Bold indicates the top two results

TABLE III
AP(%) OF PEDESTRIAN DETECTION PER DISTANCE (IOU THRESHOLD: 0.25)

Training dataset	Verification dataset	3D AP (%)			BEV AP (%)		
		$<$2.5 m	$\geq$ 2.5 m	total	$<$2.5 m	$\geq$ 2.5 m	total
KITTI-train	Sim-test	65.41	85.27	84.43	70.14	85.30	84.54
Sim-train	Sim-test	**85.56**	**88.13**	**87.84**	**85.63**	**88.14**	**87.89**

TABLE IV
F-MEASURE OF PEDESTRIAN DETECTION PER DISTANCE ON REAL DATA

Training dataset	F-measure		
	$<$2.5 m	$\geq$ 2.5 m	total
KITTI-train	30.67	68.80	63.04
Sim-train	**96.36**	**98.44**	**98.06**

tion at all the difficulty levels on KITTI. This result suggests that refection intensity helps to improve the detection accuracy when the training data and the testing data are obtained using the same LIDAR sensor and that the CosPointPillars' concept of taking the alternative feature of the reflection intensity is reasonable.

D. Comparison of dataset

To confirm that SimDataset was suitable for the pedestrian detection in a walking space, we compared the CosPointPillars trained with Sim-train and KITTI-train in pedestrian detection accuracy on both simulated and real walking spaces.

First, we compared the pedestrian detection accuracy of these two networks by using Sim-test dataset. In this comparison, the network trained with Sim-train is more advantageous because the size of the annotated bounding boxes in Sim-train is similar to that in Sim-test. Thus, to make a fair comparison, we reduced the IoU threshold to 0.25.

Table III shows the results of this experiment. In addition to the detection accuracy for the whole detection range, we also evaluated the detection accuracy for pedestrians closer than 2.5 m and the ones located more than 2.5 m to clarify the effect of Sim-train containing pedestrians within 2.5 m, which is not contained in KITTI-train. As Table III shows, in the detection of pedestrians located more than 2.5 m away, the AP difference between these two networks is less than 3% in both 3D and BEV detection. Meanwhile, in the detection of pedestrians located closer than 2.5 m, the differences in AP were 20.15% in 3D detection and 15.49% in BEV detection.

Next, we compared CosPointPillars trained with Sim-train and those trained with KITTI-train in the pedestrian detection accuracy on real data of walking spaces. In this experiment, the point clouds were obtained by a Velodyne's VLP-16 with 16 scan layers on an electric wheelchair. As a verification of the data, we randomly sampled 100 frames from one sequence of the LIDAR scans. Thereafter, we compared the F-measure of two networks in these frames under the condition of a score threshold of 0.5.

Table IV lists the F-measure of the two networks in each distance section. As shown in Table IV, the F-measures that were trained with Sim-train were improved.

Some examples of the results are shown in Fig. 6. The bounding boxes generated with scores of over 0.5 are shown. Yellow bounding boxes indicate the detection results generated by the network trained with SimDataset, whereas blue bounding boxes indicate the results from a network trained with KITTI. As shown in Fig 6, it can be seen that the network trained with SimDataset could detect more pedestrians compared with that trained with KITTI.

As these experiments show, being trained with Sim-train improved the pedestrian detection accuracy in walking spaces, particularly within close proximity.

VI. CONCLUSION

For pedestrian detection in walking spaces, we proposed CosPointPillars and SimDataset. CosPointPillars, which was independent of reflection intensity, showed higher performance particularly in BEV detection compared with the existing reflection intensity independent networks. This result showed that the CEN helped with the pedestrian detection on sparse point cloud. Furthermore, having been trained with SimDataset, CosPointPillars improved the detection performance, particularly within close proximity. This result showed that SimDataset is an appropriate dataset for walking spaces.

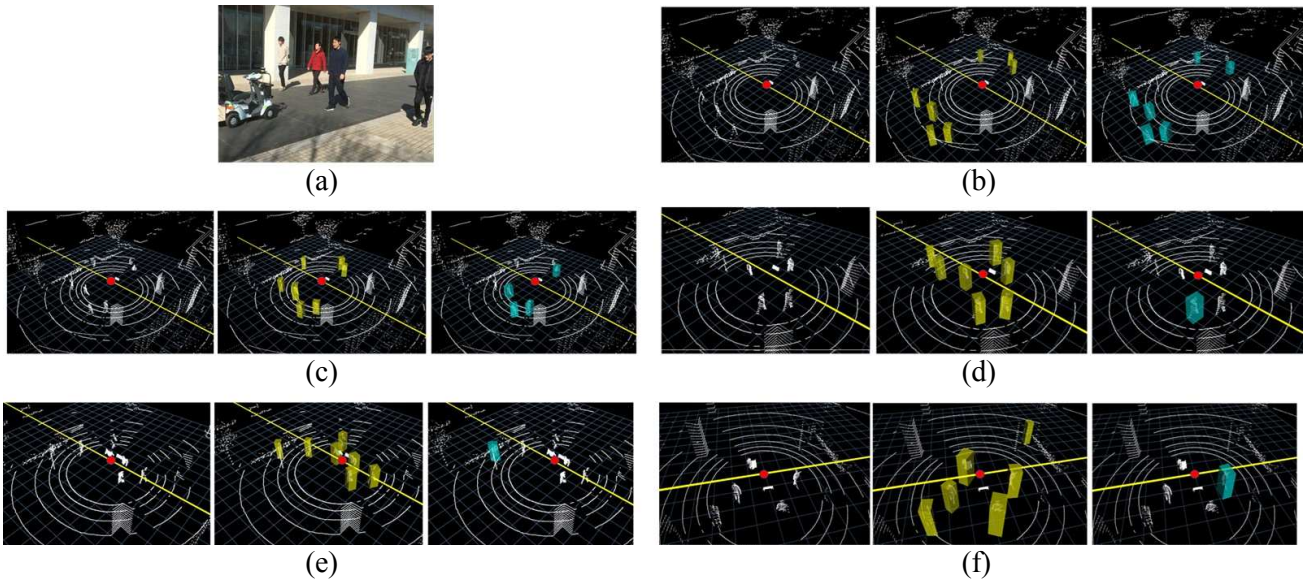

Fig. 6. Example of detected pedestrians. (a) is a snapshot of the experiment. The other figures are examples of point clouds and pedestrians detected with CosPointPillars. Five examples denoted from (b) to (f) are shown. The left figure of each example is a raw point cloud, the center figure is the point cloud and bounding boxes generated by CosPointPillars trained with SimDataset. The right figure is the point cloud and bounding boxes generated by CosPointPillars trained with KITTI. In these figures, the grid size is 1.0 m. A yellow line indicates the line of X=Y=0 and a red dot indicates the LIDAR.

Given that CosPointPillars showed a lower detection accuracy compared with original PointPillars in the experiment, future studies will include improvement of the detection accuracy of the network. To realize this, we consider that it is worthwhile improving the accuracy of the CEN.

ACKNOWLEDGMENT

We would like to thank H. Hamagaki for his efforts in the development the LIDAR simulator.

REFERENCES

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving." In CVPR, 2017.
[2] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "Joint 3D Proposal Generation and Object Detection from View Aggregation." In IROS, 2018.
[3] B. Yang, W. Luo, and R. Urtasun. "PIXOR: Real-time 3d Object Detection from Point Clouds." In CVPR, 2018.
[4] M. Simon, S. Milz, K. Amende, and H.-M. Gross. "Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds." In ECCV Workshops, 2018.
[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In CVPR, 2017.
[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "Pointnet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In NIPS, 2017.
[7] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. "Frustum Pointnets for 3d Object Detection from RGB-D Data." In CVPR, 2018.
[8] Z. Wang and K. Jia, "Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection." In IROS, 2019.
[9] Y. Zhou and O. Tuzel. "Voxelnet: End-to-end Learning for Point Cloud Based 3d Object Detection." In CVPR, 2018.
[10] Y. Yan, Y. Mao, and B. Li. "SECOND: Sparsely Embedded Convolutional Detection." Sensors, 2018. vol. 18, No. 10, p. 3337, 2018.
[11] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast Encoders for Object Detection from Point Clouds." In CVPR, 2019.
[12] A. Geiger, P. Lenz, and R. Urtasun. "Are We Ready for Autonomous Driving? the Kitti Vision Benchmark Suite." In CVPR, pp. 3354–3361, 2012.
[13] A. Patil, S. Malla, H. Gang, and Y.-T. Chen, "The H3D Dataset for Full-Surround 3D Multi-Object Detection and Tracking in Crowded Urban Scenes." In ICRA, pp. 9552–9557, 2019.
[14] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. "nuscenes: A Multimodal Dataset for Autonomous Driving." arXiv:1903.11027, 2019.
[15] S. Shi, X. Wang, and H. Li, "PointRCNN: 3d Object Proposal Generation and Detection From Point Cloud." In ICPR, 2019.
[16] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles." SPAR, vol. 5, pp. 621–635, 2018.
[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator." In CoRL, 2017.
[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y.Fu, and A. C. Berg. "SSD: Single Shot Multibox Detector." In ECCV, 2016.
[19] A. Tatoglu, and K. Pochiraju, "Point Cloud Segmentation with LIDAR Reflection Intensity Behavior." In ICRA, 2012.
[20] S. Khan, D. Wollherr, and M. Buss, "Modeling Laser Intensities For Simultaneous Localization and Mapping." RA-L, vol. 1, No. 2, pp. 692–699, 2016.
[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection." In ICCV, 2017.
[22] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman. "The Pascal Visual Object Classes (VOC) Challenge." IJCV, pp. 303–338, 2010.