ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/355578032

DNN-Based Recognition of Pole-Like Objects in LiDAR Point Clouds

Conference Paper · September 2021

DOI: 10.1109/ITSC48978.2021.9564759

CITATIONS
8

READS
377

4 authors:

Christopher Plachetka
Volkswagen AG
5 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Jenny Fricke
Technische Universität Braunschweig
5 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Marvin Klingner
Technische Universität Braunschweig
34 PUBLICATIONS **450** CITATIONS
SEE PROFILE

Tim Fingscheidt
Technische Universität Braunschweig
316 PUBLICATIONS **3,326** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Adversarial Attacks on Neural Networks View project

Multi-Task Learning for Improved Safety in DNN's View project

All content following this page was uploaded by Christopher Plachetka on 05 November 2021.

The user has requested enhancement of the downloaded file.

DNN-Based Recognition of Pole-Like Objects in LiDAR Point Clouds

Christopher Plachetka[1], Jenny Fricke[1], Marvin Klingner[2] and Tim Fingscheidt[2]

***Abstract*— In this paper, we present a novel method for recognizing pole-like objects in LiDAR point clouds, which is useful for landmark-based localization and high-definition (HD) map generation. Our method utilizes a state-of-the-art deep neural network relying on learned encodings of the point cloud input. Here, we modified an existing network architecture to improve the detection of small objects such as poles. To enable the estimation of bounding cylinders for pole-like objects, we propose a respective object anchor design with an accompanying strategy for matching ground truth objects to object anchors during network training. Furthermore, we examine the impact of two different data representations of the point cloud on the detection performance, as well as the impact of topological alternatives. The performance of our method is demonstrated on a dataset including various challenging classes of poles. We plan to publish this dataset as part of this work, which fills a gap regarding publicly available LiDAR point cloud datasets covering various elements of HD maps such as pole-like objects. Our method achieves a mean recall, precision, and classification accuracy of 0.85, 0.85, and 0.93, respectively, and may serve as a future baseline for other approaches.**

I. Introduction

The combined classification and detection of pole-like objects, which we refer to as recognition in the following, is an important task in the fields of automated driving and geodesy. Subsequently, we use the term "poles" for pole-like objects, such as trees, lampposts, traffic light poles, etc. The detection task typically includes the regression of object bounding boxes, or bounding cylinders, as in our case. In an automotive context, poles are typically detected and to be matched to high-definition (HD) map data for self-localization of the ego vehicle, which is commonly referred to as landmark-based localization [1]. Various sensor setups are utilized, such as stereo vision [2] or onboard laser scanners [3]. Obtaining the class of a pole can improve the matching result, since an additional matching criterion is available. Moreover, HD maps have to be updated continuously to ensure the safety of the driving function [4]. Thereby, the problem arises that a specific sensor setup can only detect certain classes of poles. During map updating, this may lead to the false removal of repeatedly undetected poles from the map, which could not be detected by the sensor setup. Such false deletions can be prevented by an HD map providing a pole classification, as stored poles can be marked as undetectable by a specific sensor setup.

[1]Christopher Plachetka and Jenny Fricke are with Volkswagen Group, Commercial Vehicles, Berliner Ring 2, 38440 Wolfsburg, Germany. {forename.surname}@volkswagen.de

[2]Marvin Klingner and Tim Fingscheidt are with the Institute for Communications Technology, Technische Universität Braunschweig, Schleinitzstr. 22, 38106 Braunschweig, Germany. {m.klingner, t.fingscheidt}@tu-bs.de

Fig. 1: **Recognition of pole-like objects** performed by our neural network in a high-density point cloud. Red cylinders indicate true positive detections with estimated diameters, while the gray value represents the intensity of reflected LiDAR beams. The classification result for each detection is given as red text label.

The generation of such HD maps is generally based on high-density point clouds obtained from accumulating single LiDAR scans. This is a typical task in the field of geodesy, which has come up with a multitude of algorithms to automatically extract various types of urban infrastructure objects from the point cloud to speed up the otherwise costly, manual process of map generation. Such objects include lampposts [5], [6], trees [7], or other pole classes [8], [9].

However, the recognition of poles remains a challenging task as poles have various shapes and heights, or come with a variety of attachments (e.g., traffic signs, traffic lights, or lamps). Thus, existing conventional algorithms relying on hand-crafted features do not necessarily generalize well to the variety of pole classes with differing features seen in the real world. Also, they can be computationally expensive as argued in [10], while our approach learns an optimal feature representation for various, principally generic, classes of poles in an end-to-end fashion, which makes manual feature design obsolete. In this work, we use high-density point clouds as input to our deep neural network (DNN) as visualized in Fig. 1. Given the respective training data, our pole recognition method is potentially extendable to less dense point clouds obtained from onboard laser scanners.

Our contributions to the field of pole recognition in both automotive and geodetic context are as follows. First, to the best of our knowledge, we are the first to perform a recognition of poles with a DNN-based approach using LiDAR only. Second, we compare two different data representations of the point cloud as input to the DNN. Third, we propose and examine several network topology options to improve the detection of small objects such as poles.

II. Related Work

In this section, we review state-of-the-art methods regarding pole recognition in both the automotive field and in geodesy. We conclude with a review of methods for DNN-based object detection in LiDAR point clouds.

A. Pole Recognition (Automotive Field)

In the automotive field, the dominant application of pole detection (without classification) is landmark-based self-localization. To this end, various conventional algorithms have been proposed to detect poles including a regression of position and diameter with different sensor setups for subsequent map matching and localization (e.g., LiDAR-based [1], [3]). In contrast to these methods, our method additionally provides a classification of poles and may be integrated into existing end-to-end learning pipelines (e.g., [11], [12]) for perceiving moving objects in the environment.

B. Pole Recognition (Geodesy)

In geodesy, a wide variety of methods exist for pole recognition, which typically operate on high-density point clouds only. Such methods utilize different approaches for detection and classification in combination, which we group into shape-based [5], [13]–[15], feature-based [5], [6], [8]–[10], and machine learning-based [8], [16]–[18] approaches.

Shape-based methods rely on prior assumptions regarding the shape of the objects to be detected or classified. For instance, cylindrical [13] or circular [14] shapes are fitted to the voxelized point cloud to detect poles. The classification of poles can be obtained using 3D shape matching [5], [15] or a predefined set of rules, e.g., related to possible heights of specific pole classes [14].

Feature-based approaches extract features such as geometrical [6], intensity [6], or density [9] characteristics from the point cloud as input to a subsequent pole detection or classification. Downstream detection or segmentation algorithms either rely on heuristic criteria [8], [10], e.g., the isolation of a pole with respect to its local surroundings [10], or on common algorithms applied in computer vision such as graph-cut [5], [6] and Euclidean clustering [5], [6].

Downstream classification approaches (e.g., classifying previously obtained point cloud segments) are typically employing machine learning, e.g., Gaussian mixture models [6], support vector machines [8], [16], [17], or convolutional neural networks (CNNs) [18]. However, compared to the aforementioned approaches, our method combines detection and classification in one holistic approach, extends easily to different pole classes (given enough training data), and is capable to learn complex features to handle challenges such as (partly) occluded poles or various pole attachments without the need to adopt specific further algorithms.

C. DNN-Based Object Detection in LiDAR Point Clouds

Early works for DNN-based object detection in point clouds rely on computationally expensive 3D convolutions performed on voxelized point clouds [19], [20] with hand-crafted feature encodings. Enabled by the seminal work of Charles *et al.* [21], the encoding paradigm has shifted towards learned feature encodings [11], [12], [22], which reduce the loss of geometrical information due to the discretization of the point cloud. Furthermore, recent research focuses on point-based methods [23], [24], entirely omitting the discretization step. However, as our pole recognition method serves as an intermediate step towards map deviation detection [4] based on overlaying map and sensor data in a discretized bird's eye view, we select the `PointPillars` architecture [11] that allows for such input. However, as the architecture is designed to detect rather large objects such as cars, we apply modifications to the architecture to improve the detection of smaller objects, i.e., poles.

III. DNN-Based Pole Recognition Method

In this section, we introduce our DNN-based method for recognizing poles in LiDAR point clouds. First, we present the architecture of the utilized network. Then, we introduce our object anchor design with a respective strategy for matching ground truth objects to anchors during training. Finally, we define the loss functions used to train the network.

A. Network Overview

The network used in this paper is a modified version of the `PointPillars` architecture by Lang *et al.* [11], which is visualized in Fig. 2. The network contains an encoder stage for LiDAR point clouds, learned in an end-to-end fashion, a backbone for further feature extraction, and two output heads: one classification head yielding each pole class individually, and one regression head estimating bounding cylinders. Subsequently, we provide detailed explanations for each of the network parts.

B. Encoder

Let $\boldsymbol{v} \in \mathcal{V}$ be a single LiDAR point from the LiDAR point cloud $\mathcal{V}$, with $\mathcal{V}$ being an unordered set of LiDAR points. Each LiDAR point $\boldsymbol{v} = (v^{\mathrm{int}}, \boldsymbol{v}^{\mathrm{crd}})$ contains an intensity measurement $v^{\mathrm{int}} \in \mathbb{I}$, with $\mathbb{I} = [0, 1]$, and a coordinate measurement $\boldsymbol{v}^{\mathrm{crd}} \in \mathbb{R}^3$ in Cartesian space. In a first processing step to obtain a learned encoding for $\mathcal{V}$, each LiDAR point $\boldsymbol{v}$ is sorted into a spatial bird's eye view grid with N^{x} and N^{y} cells in the x- and y-dimension of the grid, respectively. While sorting points into cells, we allow only a maximum number of K points per cell, using the setting $K = 100$ as applied in the original architecture [11]. If a cell contains more than K points, random sub-sampling is applied. From this spatial grid, only $N \leq N^{\mathrm{x}} \cdot N^{\mathrm{y}}$ populated cells are considered for the further encoder stage. We reduce runtime and network size both during training and inference by creating another grid of size $N \times K$, containing only occupied cells and their respective points. These populated cells are indexed by $n \in \mathcal{N} = \{1, ..., N\}$.

Subsequently, each LiDAR point $\boldsymbol{v} \in \mathcal{V}_n \subset \mathcal{V}$ in a populated cell n, with $\mathcal{V}_n$ being the subset of points in cell n, is augmented with the mean $\overline{\boldsymbol{v}}_n \in \mathcal{V}_n$ of all points of that cell, and with the cell center coordinates $\boldsymbol{w}_n \in \mathbb{R}^2$ in the x-y-plane, yielding a total of nine features for each point

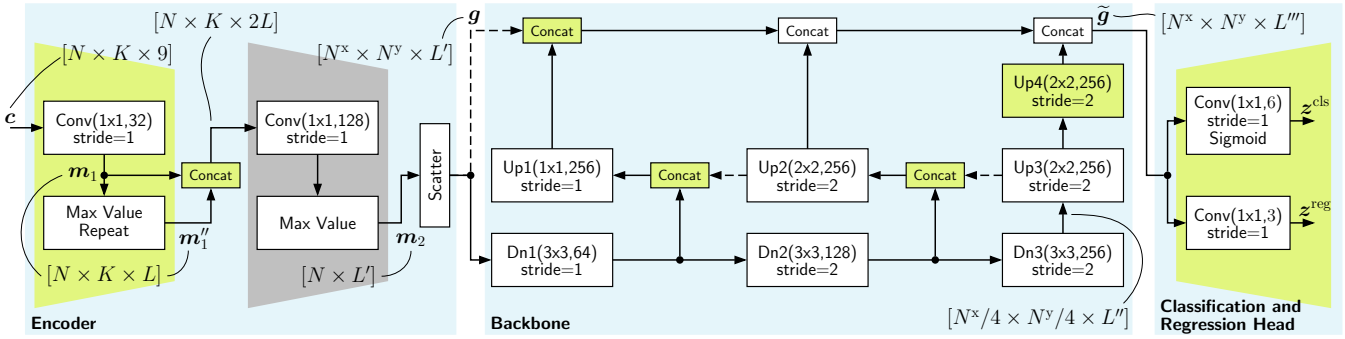


Fig. 2: **Architecture of the DNN**. Blue: Processing stages in the network. Green: Modifications applied to the original architecture of `PointPillars` [11]. The brackets $(m \times m,\ F)$ indicate the kernel size $m \times m$ and the number of filters F used in an ordinary or transposed convolution. "Concat" blocks indicate a concatenation operation. Down (Dn) blocks comprise multiple convolutional layers. We use a final encoding depth of $L' = 128$ features. In Up3, the original architecture uses a kernel size of 4×4 with a stride of 4.

in the grid of size $N \times K$, summarized as "point feature vector" $\boldsymbol{x}_{n,k} \in \mathcal{X}_{n,k} = \{v_{n,k}^{\mathrm{int}}, \boldsymbol{v}_{n,k}^{\mathrm{crd}}, \overline{\boldsymbol{v}}_n, \boldsymbol{w}_n \,|\, v_{n,k}^{\mathrm{int}} \in \mathbb{I}, \boldsymbol{v}_{n,k}^{\mathrm{crd}} \in \mathbb{R}^3, \overline{\boldsymbol{v}}_n \in \mathbb{R}^3, \boldsymbol{w}_n \in \mathbb{R}^2\}$, with $k \in \mathcal{K} = \{1, ..., K\}$. A dense cell tensor $\boldsymbol{c} \in \mathcal{X}_{n,k}^{N \times K}$ is then obtained, containing $N \times K$ point feature vectors. If a cell contains less than K points, zero-padding is applied. During this process, a lookup table is created that maps a cell index n to its original position in the spatial grid with $N^{\mathrm{x}} \times N^{\mathrm{y}}$ cells for later usage. The dense cell tensor $\boldsymbol{c}$ provides a total of nine $(N \times K)$-dimensional feature maps as input to the encoder (see tensor $\boldsymbol{c}$ in Fig. 2).

The encoder network used in `PointPillars` [11] is based on `PointNet` [21] and has the objective to compute one (abstract) feature vector for each of the N grid cells by encoding all K point feature vectors contained in a cell. Subsequently, we term such a feature vector as "cell feature vector". Following `VoxelNet` [12], we add a second encoder stage to the `PointPillars` architecture (see the left green parts in Fig. 2) to improve the representational power of the cell feature vectors.

As a first step to obtain cell feature vectors, all points $k \in \mathcal{K}$ of all cells $n \in \mathcal{N}$ contained in $\boldsymbol{c} = (\boldsymbol{x}_{n,k})$ are jointly mapped to a feature space with $L = 32$ abstract features in the first encoder stage. To achieve this, a 1x1 convolution with 32 filters is performed on $\boldsymbol{c}$, which results in the feature tensor $\boldsymbol{m}_1 \in \mathbb{R}^{N \times K \times L}$ providing K feature vectors of length L for each of the N cells. Now, to obtain cell feature vectors from the feature tensor $\boldsymbol{m}_1$, the point dimension K must be reduced to 1 by applying the symmetric $\max$ operation to $\boldsymbol{m}_1$. For each cell, this operation provides one maximum value for each of the 32 encoded features among all K points contained in that cell. Thus, the resulting feature tensor $\boldsymbol{m}_1' \in \mathbb{R}^{N \times 1 \times L}$ of reduced size (calculated in the "Max Value Repeat" block in Fig. 2) provides one cell feature vector of length L for each of the N cells, which could already be used as an input to the backbone.

However, these cell feature vectors encode the entire group of points contained in that cell, but are not yet able to encode interactions between points within this group, which is due to the $\max$ operation combining the encodings of all K points in a cell. To obtain cell feature vectors with greater representational power, encoded feature vectors obtained point-wise (contained in $\boldsymbol{m}_1$) must be concatenated with cell feature vectors (contained in $\boldsymbol{m}_1'$) that encode the local group of points. To this end, the cell feature vectors contained in $\boldsymbol{m}_1'$ are repeated K times to match dimensions with $\boldsymbol{m}_1$, which yields $\boldsymbol{m}_1'' \in \mathbb{R}^{N \times K \times L}$ (see the "Max Value Repeat" block in Fig. 2). Subsequently, the feature tensor $\boldsymbol{m}_1''$ is concatenated with $\boldsymbol{m}_1$ in the L-dimension to provide the input for the second encoder stage, which repeats the processing steps of the first encoder stage, while omitting both the concatenation and the repeat operation (using the $\max$ operation only). Using $L' = 128$ filters for the second 1x1 convolution, the final feature tensor $\boldsymbol{m}_2 \in \mathbb{R}^{N \times L'}$ is obtained, containing the final N cell feature vectors.

In a last processing step of the encoder, the cell feature vectors in $\boldsymbol{m}_2$ are scattered back to their original positions in the spatial grid of size $N^{\mathrm{x}} \times N^{\mathrm{y}}$ using the lookup table generated during the creation of $\boldsymbol{c}$, which yields the encoded spatial grid $\boldsymbol{g} \in \mathbb{R}^{N^{\mathrm{x}} \times N^{\mathrm{y}} \times L'}$ as input to the backbone, with zero-padding applied to empty cells.

C. Backbone

The backbone stage of the network processing $\boldsymbol{g}$ in Fig. 2 is composed of a downstream and an upstream network. As the inclusion of more context is beneficial for detecting small objects [25], we modified the architecture, i.e., by adding a feature pyramid[1] [26], which concatenates low-level features from the downstream path with high-level features from the upstream path (green concatenation operations between "up" blocks in Fig. 2 with new dashed line inputs). The downstream network contains three blocks ("dn" in Fig. 2), containing 3, 5, and 5 layers, respectively. Each layer is composed of one convolution, one batch normalization, and one ReLU activation. Upsampling blocks each contain one transposed convolution, followed by one batch normalization and one ReLU activation. Using the upstream network, feature maps originating from all three stages of the downstream network are eventually concatenated to provide more fine-grained features to the network heads (concatenation operations at the top of Fig. 2). Compared to the original feature

[1]We substitute the addition operation used in the original paper [26] with the concatenation operation, which proofed to be beneficial for our use case.

pyramid architecture [26], we additionally concatenate $\boldsymbol{g}$ (dashed line) to features extracted by the backbone to further enrich the representational power of the grid with features that are obtained per cell. With the backbone providing $3L'' = 3 \cdot 256 = 768$ features, and $\boldsymbol{g}$ providing $L' = 128$ features, the concatenations (top of Fig. 2) yield a backbone output $\widetilde{\boldsymbol{g}} \in \mathbb{R}^{N^{\mathrm{x}} \times N^{\mathrm{y}} \times L'''}$ with $L''' = 3L'' + L' = 896$.

D. Classification and Bounding Cylinder Regression Head

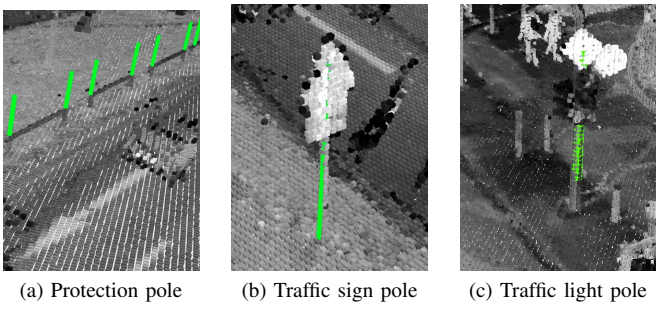
Subsequently, we refer to the grid-like outputs of the network heads as "predictions". Using a post-processing step, these predictions are converted into a list of "object estimations", which comprise an existence likelihood (commonly referred to as "objectness score"), and estimated bounding cylinder parameters (position and diameter). The network operates in a single-shot fashion [27] to compute predictions. Here, to facilitate the detection task, the network predicts existence likelihoods using the classification head for a predefined set of so-called object anchors (referred to as "anchor" in the following), which are default-sized objects placed in the output grid. If an anchor is likely to contain an object, the objectness score increases. Additionally, the network predicts bounding cylinders using the regression head by adopting the object anchors to match a detected object's position and diameter. Subsequently, we define the anchor grid, the classification output $\boldsymbol{z}^{\mathrm{cls}}$, and the regression output $\boldsymbol{z}^{\mathrm{reg}}$ of the network heads (cf. Fig. 2).

Anchors are placed in a 2D grid with the same range and number of cells as the encoded spatial grid $\boldsymbol{g}$ of size $G = N^{\mathrm{x}} \cdot N^{\mathrm{y}}$ without downsampling to facilitate the resolution of small objects, with a single anchor being placed in each cell. We denote $g \in \mathcal{G}$ as the index of the 2D grid, with $\mathcal{G} = \{1, 2, ..., G\}$ being the set of possible grid cell indices. Accordingly, predictions for single cells $\boldsymbol{z}_g^{(\cdot)}$ (either being predicted objectness scores or predicted regression values) are contained in one tensor $\boldsymbol{z}^{(\cdot)} = \left(\boldsymbol{z}_1^{(\cdot)}, ..., \boldsymbol{z}_G^{(\cdot)}\right)$.

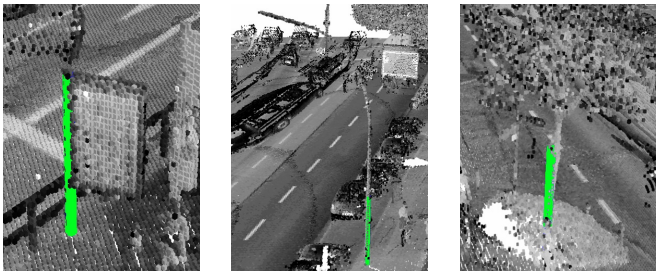
The classification head comprises one convolutional layer that detects each pole class individually. To this end, the classification head predicts multiple class-specific objectness scores for each object anchor, contained in the head's output $\boldsymbol{z}^{\mathrm{cls}} = (\boldsymbol{z}_{g,6}^{\mathrm{cls}}) \in \mathbb{I}^{G \times 6}$, with $\mathbb{I} = [0, 1]$ and $\mathcal{S} = \{\texttt{protection}, \texttt{sign}, \texttt{light}, \texttt{billboard}, \texttt{lamppost}, \texttt{tree}\}$[2] being the set of six classes visualized in Fig. 3. Note that no separate class is designated to the background. Single class-specific scores for an anchor contained in cell g are denoted $z_{g,s}^{\mathrm{cls}} \in \mathbb{I}$ with $s \in \mathcal{S}$ being the class index.

The bounding cylinder regression head predicts respective regression values for detected poles using a second convolutional layer. Such regression values are predicted for each object anchor, but are only considered for further evaluation if the anchor actually contains an object (indicated by the class-specific objectness scores $z_{g,s}^{\mathrm{cls}}$). We denote the set of bounding cylinder parameters (either belonging to an estimated object O or to an anchor A) as $\mathcal{Q} = \{x^{\mathrm{pos}}, y^{\mathrm{pos}}, d\}$,

[2]Given class names are abbreviations for: protection pole, traffic sign pole, traffic light pole, and billboard pole.



(a) Protection pole (b) Traffic sign pole (c) Traffic light pole



(d) Billboard pole (e) Lamppost (f) Tree

Fig. 3: **Examples for pole classes** contained in our dataset. Ground truth bounding cylinders are colored in green.

with x^{pos} and y^{pos} being the cylinder's coordinates in the x-y-plane, and d being the cylinder's diameter. Desired bounding cylinder parameters $q_g^{\mathrm{O}} \in \mathcal{Q}_g^{\mathrm{O}} = \{x_g^{\mathrm{pos,O}}, y_g^{\mathrm{pos,O}}, d_g^{\mathrm{O}}\}$ of an object O are not estimated directly by cell g. Instead, to facilitate the regression task, the network adopts the position and diameter of an anchor A contained in cell g to match the respective values of an estimated object O. To this end, the network predicts normalized "anchor offsets" $\boldsymbol{z}_g^{\mathrm{reg}} = (z_g^{(x^{\mathrm{pos}})}, z_g^{(y^{\mathrm{pos}})}, z_g^{(d)})$, which indicate the difference in position and diameter between the object's estimated parameters $(x_g^{\mathrm{pos,O}}, y_g^{\mathrm{pos,O}}, d_g^{\mathrm{O}})$ and default anchor parameters $(x_g^{\mathrm{pos,A}}, y_g^{\mathrm{pos,A}}, d_g^{\mathrm{A}})$. For the similar definition of the positional offsets $z_g^{(x^{\mathrm{pos}})}$ and $z_g^{(y^{\mathrm{pos}})}$, we summarize both offsets as $z_g^{(p)} \in \{z_g^{(x^{\mathrm{pos}})}, z_g^{(y^{\mathrm{pos}})}\}$, and the respective positional cylinder parameters as $p \in \{x^{\mathrm{pos}}, y^{\mathrm{pos}}\}$. With $r_{\mathrm{cell}} \times r_{\mathrm{cell}}$ being the size of the square-shaped grid cells (e.g., $r_{\mathrm{cell}} = 0.2\,\mathrm{m}$ in this paper), we can now define

$$z_g^{(p)} = \frac{(p_g^{\mathrm{O}} - p_g^{\mathrm{A}})}{r_{\mathrm{cell}}}, \quad p \in \{x^{\mathrm{pos}}, y^{\mathrm{pos}}\}, \tag{1}$$

$$z_g^{(d)} = \log\Big(\frac{d_g^{\mathrm{O}}}{d_g^{\mathrm{A}}}\Big), \quad d \in \mathbb{R}^+. \tag{2}$$

During inference, this normalization has to be reverted to obtain the bounding cylinder parameters $q_g^{\mathrm{O}} \in \mathcal{Q}_g^{\mathrm{O}}$ of a detected pole. Thereby, only anchors with index g that have a predicted classification score

$$\max_{s \in S}(z_{g,s}^{\mathrm{cls}}) \geq \Theta^{\mathrm{score}} \tag{3}$$

are considered as detection, with the threshold Θ^{score} being determined in Section V. Object estimations with overlapping bounding cylinders originating from multiple anchors but estimating the same real-world object are filtered using non-maximum-suppression [11], [22] to obtain the final object

list. Instead of using intersection over union (IoU) to measure the overlap of box predictions [11], [22], we require the minimum distance between predicted box positions to be $0.3\,\mathrm{m}$, below which object estimations are suppressed in such a way that only the prediction with the highest score remains.

E. Anchor Design and Matching Strategy

In this work, we use a simple anchor design featuring one single-sized anchor located in the center of each grid cell g with a diameter of $d_g^{\mathrm{A}} = 0.2\,\mathrm{m}$, which equals the cell size r_{cell} used in this paper and corresponds approximately to the mean diameter of all poles contained in our dataset.

During training, ground truth objects have to be matched to anchors to generate the target vectors $\overline{\boldsymbol{z}}_g^{\mathrm{cls}}$ and $\overline{\boldsymbol{z}}_g^{\mathrm{reg}}$, with the overline denoting ground truth. If an anchor contained in cell g matches with the ground truth object, we set the anchor's classification target $\overline{z}_{g,\overline{s}}^{\mathrm{cls}} = 1$ according to the true object's class $\overline{s}$. The regression target vector $\overline{\boldsymbol{z}}_g^{\mathrm{reg}}$ (anchor offsets) is being set according to the parameterization defined by (1) and (2), using the respective ground truth cylinder parameters $(\overline{x}_g^{\mathrm{pos,O}}, \overline{y}_g^{\mathrm{pos,O}}, \overline{d}_g^{\mathrm{O}})$ for $(x_g^{\mathrm{pos,O}}, y_g^{\mathrm{pos,O}}, d_g^{\mathrm{O}})$. To ensure that a ground truth object is matched with at least one anchor, we consider the anchor closest to the object as match (see left matching anchor in Fig. 4). Additionally, to measure the overlap with anchors surrounding the ground truth object, we define the "intersection over smaller area" (IosA) measure as the intersecting area of the two circles in the x-y-plane (given by a respective anchor and the ground truth cylinder diameter), indicated relative to the smaller of both circle areas[3]. We require $\mathrm{IosA} > 0.2$ to consider surrounding anchors as matches[4], whereby multiple anchors can be matched with a single ground truth object (see the three matching anchors visualized as dashed circles on the right of Fig. 4).

Otherwise, if $\mathrm{IosA} = 0$, an anchor is considered as background (anchor not containing an object) with $\overline{z}_{g,s}^{\mathrm{cls}} = 0$ and $\overline{\boldsymbol{z}}_g^{\mathrm{reg}} = \mathbf{0}$. Anchors that cannot be clearly classified as match or (unmatched) background with $0 < \mathrm{IosA} < 0.2$ (referred to as "don't care" state in the following), are not penalized during loss computation to facilitate the training process using a mask factor $\lambda_g^{\mathrm{mask}} \in \{0, 1\}$, which equals 0 in the "don't care" state and 1 otherwise.

F. Loss Function Definition

Similar to `PointPillars` [11] and `SECOND` [22], we define the loss function optimizing the network as

$$J = \frac{1}{N^{\mathrm{poles}}} \cdot \sum_{g \in \mathcal{G}} \lambda \cdot J_g^{\mathrm{cls}}(\overline{\boldsymbol{z}}_g^{\mathrm{cls}}, \boldsymbol{z}_g^{\mathrm{cls}}) + (1-\lambda) \cdot J_g^{\mathrm{reg}}(\overline{\boldsymbol{z}}_g^{\mathrm{reg}}, \boldsymbol{z}_g^{\mathrm{reg}}), \tag{4}$$

with N^{poles} being the number of poles contained in a training sample, $\lambda = 2/3$ being a weighting factor, and J_g^{cls} and J_g^{reg} being the classification and regression loss, respectively. The classification loss J_g^{cls} is formulated as focal loss [28] to improve the performance for objects that are particularly hard to classify [11]. Using the distinction of cases depending on $\overline{z}_{g,s}^{\mathrm{cls}}$ in (5), a higher weight is assigned to cells for which the prediction is further away from the target value. Using the original paper settings for the weight factors $\alpha = 0.25$, $\beta = 2$, and with $\mathcal{S} = \{\texttt{protection}, \texttt{sign}, \texttt{light}, \texttt{billboard}, \texttt{lamppost}, \texttt{tree}\}$ and $\lambda_g^{\mathrm{mask}} \in \{1, 0\}$ (explained in Section III-E), we define:

$$J_g^{\mathrm{cls}} = \lambda_g^{\mathrm{mask}} \sum_{s \in \mathcal{S}} -\alpha_{g,s} \cdot (1 - \gamma_{g,s})^{\beta} \cdot \log(\gamma_{g,s}), \tag{5}$$

$$\gamma_{g,s} = \begin{cases} z_{g,s}^{\mathrm{cls}} & \text{if class } s \text{ in cell } g \ (\overline{z}_{g,s}^{\mathrm{cls}} = 1) \\ 1 - z_{g,s}^{\mathrm{cls}} & \text{otherwise} \end{cases},$$

$$\alpha_{g,s} = \begin{cases} \alpha & \text{if class } s \text{ in cell } g \ (\overline{z}_{g,s}^{\mathrm{cls}} = 1) \\ 1 - \alpha & \text{otherwise} \end{cases}.$$

The regression loss is formulated using the smooth L1-loss (also known as Huber loss [29]). With $q \in \mathcal{Q} = \{x^{\mathrm{pos}}, y^{\mathrm{pos}}, d\}$ being a bounding cylinder parameter and $\Delta z_g(q) = z_g^{(q)} - \overline{z}_g^{(q)}$ being the difference of vector elements contained in $\boldsymbol{z}_g^{\mathrm{reg}}$ and $\overline{\boldsymbol{z}}_g^{\mathrm{reg}}$ (prediction output and target vector for cell g, respectively, see Section III-D for definition of $z_g^{(q)}$ vector elements), and with $\lambda_g^{\mathrm{obj}} \in \{1, 0\}$ being 1 only if an object is present in that cell, we obtain:

$$J_g^{\mathrm{reg}} = \lambda_g^{\mathrm{obj}} \sum_{q \in \mathcal{Q}} \mathrm{smoothL1}(\Delta z_g(q)), \tag{6}$$

$$\mathrm{smoothL1}(\Delta z_g) = \begin{cases} 0.5(\Delta z_g)^2 & \text{if } |\Delta z_g| \leq 1 \\ |\Delta z_g| - 0.5 & \text{otherwise} \end{cases}. \tag{7}$$

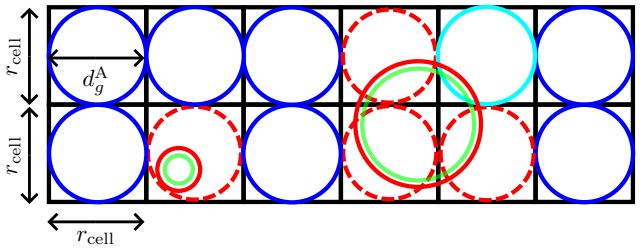


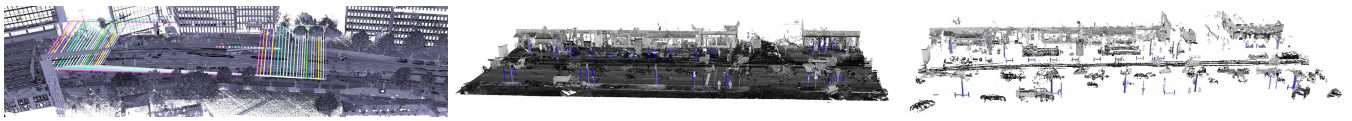
Fig. 4: **Strategy** for matching ground truth objects (green) to anchors. Matching anchors (dashed circles) have their target anchor offsets $\overline{\boldsymbol{z}}_g^{\mathrm{reg}}$ set to match the ground truth object's position and diameter (visualized as the three overlapping, solid red circles on the right, yielding only one visible solid red circle), with the diameter being slightly increased for better visibility. Unmatched background anchors (blue) keep their default size. "Don't care" anchors (cyan) feature an insufficient overlap with the ground truth object.

IV. Experimental Setup

In this section, we describe our experimental setup. First, we present the dataset used in this work. Second, we provide our definition of the evaluation metrics. We close with a description of the training procedure.

[3]The IosA measure provides a maximum value of 1 if the anchor completely contains the ground truth object or vice versa.

[4]Threshold values for IosA have been determined empirically.



(a) Sample generation (b) Dense representation (c) Sparse representation

Fig. 5: **Generation of point cloud samples in different data representations**. Point cloud samples are generated along a recorded real-world trajectory with their origin and extent being visualized as colored points and rectangles, respectively (a). Compared to the dense representation (b) of such a sample, the sparse representation (c) excludes ground points.

TABLE I: **Dataset**. Available poles refer to the overall number of unique poles provided by the dataset. The accumulated number of poles refers to multi-occurrences of single poles in multiple samples and is further split into a training, validation, and test set.

Pole Class	Available	Accumulated	Training	Validation	Test
`protection`	24384	7,062,257	3,008,060	1,817,062	2,237,135
`sign`	6620	1,874,425	791,710	492,405	590,310
`light`	2267	769,121	327,331	203,163	238,627
`billboard`	345	100,504	39,510	29,152	31,842
`lamppost`	3566	978,966	406,033	268,432	304,501
`tree`	9862	2,470,141	1,041,141	655,554	773,446

A. Dataset

As no public dataset containing pole labels for LiDAR data is available, we train and evaluate our network on a self-generated dataset that is planned to be published, containing a large high-density point cloud (HDPC) covering approx. $127\,\mathrm{km}$ of road sections[5] in the inner city of Hamburg, Germany. This type of LiDAR data is typically used to generate high-definition (HD) maps for automated driving. For our experiments, we downsampled the HDPC to a point density of $1/(10\,\mathrm{cm}^3)$ to reduce the dataset size. To mimic onboard LiDAR scans, we generate samples by taking crops from the large HDPC based on real-world trajectories recorded during a measurement campaign [4], with a minimum distance of $1\,\mathrm{m}$ between samples, as shown in Fig. 5. We defined a rectangular range of the crop ($x_{\min} = -10\,\mathrm{m}$, $x_{\max} = 70\,\mathrm{m}$, $y_{\min} = -20\,\mathrm{m}$, $y_{\max} = 20\,\mathrm{m}$, $z_{\min} = -4\,\mathrm{m}$, $z_{\max} = 4\,\mathrm{m}$) that extends more towards the vehicle's front to particularly focus on the detection of poles ahead of the vehicle. The z-dimension is normalized by subtracting the mean of all ground-classified points. We consider two data representations of the point cloud sample (see Fig. 5). The dense representation preserves all points within the crop, while the sparse representation excludes ground-classified points to reduce the number of populated cells N, reducing both network size and training time.

Table I summarizes the size of our dataset. In total, our dataset contains 89,346, 17,102, and 33,474 point cloud samples in the training, validation, and test set, respectively. Due to the cropping of samples from a large HDPC, the same poles occur in various samples at different positions and orientations, while we ensure that an individual pole does not appear in different subsets. These multi-occurrences are counted as individual elements to be detected.

[5]A road section groups all parallel lanes with the same driving direction, regardless the number of parallel lanes. A "road" on the other hand comprises a maximum of two road sections with opposing driving directions.

B. Metrics

The network's detection performance is evaluated irrespective of the classification's correctness using:

$$\text{Recall:} \qquad \mathrm{RE} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{8}$$

$$\text{Precision:} \qquad \mathrm{PR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{9}$$

$$\mathrm{F}_1\ \text{score:} \qquad \mathrm{F}_1 = 2 \cdot \frac{\mathrm{PR} \cdot \mathrm{RE}}{\mathrm{PR} + \mathrm{RE}}, \tag{10}$$

with TP and FP being the number of true and false positives, respectively, and FN being the number of false negatives. Given a correct detection, the classification task is evaluated independently using the accuracy measure

$$\mathrm{ACC} = \frac{N}{\overline{N}}, \tag{11}$$

with N being the number of correctly classified poles, and $\overline{N} = \mathrm{TP}$ being the number of correctly detected poles in the test set. Furthermore, to evaluate the estimation of a bounding cylinder parameter $\boldsymbol{q}$, we define the L1 or L2 regression error measure

$$E(\boldsymbol{q}) = \begin{cases} \frac{1}{\mathrm{TP}} \sum_{i=1}^{\mathrm{TP}} ||\boldsymbol{q}_i - \overline{\boldsymbol{q}}_i||_2 & \text{if } \boldsymbol{q} = \boldsymbol{p} \\ \frac{1}{\mathrm{TP}} \sum_{i=1}^{\mathrm{TP}} ||\boldsymbol{q}_i - \overline{\boldsymbol{q}}_i||_1 & \text{otherwise} \end{cases}, \tag{12}$$

with $\boldsymbol{q} \in \{\boldsymbol{p}, d\}$, $\boldsymbol{p} = (x^{\mathrm{pos}}, y^{\mathrm{pos}})$ being an estimated pole's center position given by x^{pos} and y^{pos}, and i being the index of the estimated object.

C. Training Procedure

Our experimental pipeline is implemented in `PyTorch`. For our experiments, we use a grid size of $N^{\mathrm{x}} = 800$ and $N^{\mathrm{y}} = 400$, which yields a grid cell size of $r_{\mathrm{cell}} \times r_{\mathrm{cell}}$, with $r_{\mathrm{cell}} = 0.2\,\mathrm{m}$. Our network is trained for 5 epochs using a 2-GPU setup (`Tesla V100`) with a batch size of 2. We use the Adam optimizer with a fixed learning rate of $2 \cdot 10^{-4}$ and a momentum of 0.9. Furthermore, we apply a global augmentation, whereby we apply a random translation $\boldsymbol{t} \in \mathcal{N}^2(\mu = 0, \sigma = 1)$, with $\mathcal{N}()$ being the Gaussian distribution, and random rotation around the z-axis by $\varphi \in \mathcal{U}$, with $\mathcal{U} = [-20°, 20°]$, to both the point cloud crop and the poles contained in that crop. To address overfitting due to multi-occurrences of individual poles in different point cloud samples, we randomly remove 10% of all points in each sample.

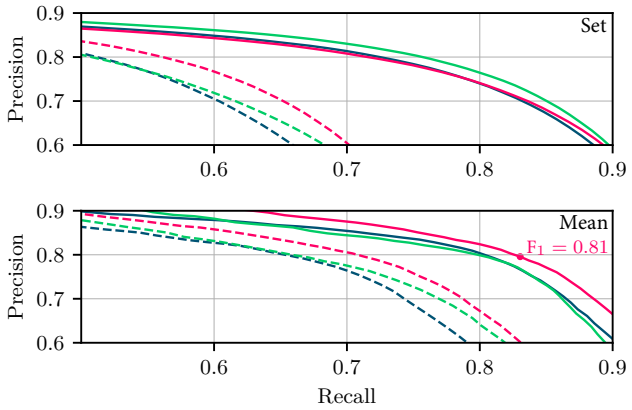


Fig. 6: **Precision-recall curves** on the **validation set** for the "set" (top) and the "mean" evaluation method (bottom). The data representations are indicated using dashed (sparse representation) or solid lines (dense representation). Different network configurations are color-coded: Blue: "baseline", green: "second stage", red: "full".

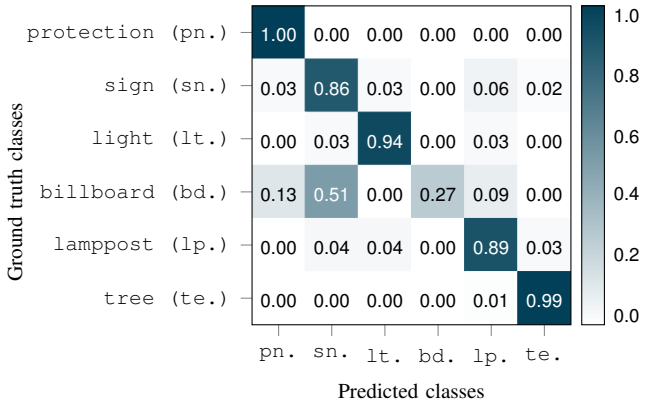


Fig. 7: **Test set confusion matrix** for the selected setup ("full" configuration using the dense representation).

V. Experimental Results

In this section, we present and discuss the experimental results for our pole recognition method. First, we examine the effect of both data representations and network topologies on the detection performance. Second, we evaluate the best performing setup in greater detail.

A. Data Representations and Network Topology Choices

We consider three network configurations, which we test with both data representations visualized in Fig. 5. The "baseline" configuration excludes all green network parts in Fig. 2 using only a single encoder stage with $L' = 64$ as in the original `PointPillars` architecture [11], while "second stage" includes the second encoder stage, and "full" represents our entire proposed network topology as shown in Fig. 2. To evaluate the detection performance on the validation set, we create the precision-recall curves in Fig. 6 by varying the threshold Θ^{score} (see (3)) in steps of 0.01, above which anchors are considered to contain an object. First, we consider the "set" evaluation method (top of Fig. 6), where only one precision and recall value is computed for the entire set of poles. Second, we use the "mean" method (bottom of Fig. 6) by calculating class-specific values for recall and precision, which are subsequently averaged. Results for the "set" method are dominated by the most common "protection pole" class (cf. Table I), while the "mean" method provides class-balanced results. Poles are only counted as TP if the Euclidean distance between predicted and ground truth cylinder center is smaller than $0.3\,\text{m}$.

Fig. 6 reveals that the dense representation, which allows the network to learn more context from the scene, is superior to the sparse representation for both evaluation methods. Regarding the "set" evaluation method, the "second stage" configuration shows the best performance, which is due to an improved detection of the most common "protection pole" class. However, this class is the least significant regarding landmark-based localization, as respective poles are particularly small and hard to detect. Thus, regarding the "mean" evaluation method, the "full" configuration is identified as the best-performing setup. The increased performance, at the cost of an increased runtime, is attributed to the feature pyramid incorporating additional context, which improves the detection of other pole classes that feature attachments (e.g., signs) or tree crowns. The best-performing setup is suitable for semi-automatic map generation, while the faster "baseline" configuration using the sparse representation may be considered for an onboard pole recognition. We select the operating point of the best-performing setup by maximizing the F_1 score for the "mean" evaluation method in Fig. 6, which yields $\text{F}_1 = \text{F}_{1,\text{max}} = 0.81$ using the threshold $\Theta^{\text{score}} = 0.25$ for further analysis on the test set.

B. Test Set Analysis

Table II shows the results obtained on the test set for the selected operating point. The best detection performance is achieved for traffic light poles ($\text{F}_1 = 0.97$) and lampposts ($\text{F}_1 = 0.94$), which provide consistent features among various samples, in contrast to traffic sign poles, featuring various attachments, for instance. Also, trees are detected well ($\text{F}_1 = 0.84$) due to their comparatively large size. The low recall ($\text{F}_1 = 0.67$) for small protection poles may be countered by decreasing Θ^{score}, or by including a size-based weight factor into the loss (5). Regarding the classification task, best performance is achieved for protection poles ($\text{ACC} = 1.00$), trees ($\text{ACC} = 0.99$), and traffic light poles ($\text{ACC} = 0.97$). Only billboard poles are classified ($\text{ACC} = 0.27$) and detected ($\text{F}_1 = 0.39$) unreliably, as they are underrepresented in the dataset (cf. Table I) and easily confused with traffic sign poles (see Fig. 7). Thus, we exclude them as invalid class for the concluding performance evaluation (last row of Table II) to obtain more realistic results. Position and diameter errors are similar among all classes, yielding mean values of $E(\boldsymbol{p}) = 4.7\,\text{cm}$ and $E(d) = 3.8\,\text{cm}$, respectively. To obtain an initial assessment of our method's performance in comparison to methods using HDPCs only published in geodesy, we present reported results in Table III. Utilized HDPC datasets are not public and differ from our dataset, which prevents a fair comparison. However, our method achieves results similar to the presented ones.

TABLE II: **Test set results** for our selected setup ("full" configuration with dense representation). The last row indicates the class mean without the billboard (bd.) class.

Class	RE	PR	F_1	ACC	$E(\boldsymbol{p})$ [cm]	$E(d)$ [cm]
Set	0.76	0.79	0.78	0.96	4.7	3.6
protection	0.67	0.76	0.71	**1.00**	4.1	2.5
sign	0.80	0.78	0.79	0.86	5.6	2.5
light	**0.98**	**0.96**	**0.97**	0.94	3.6	3.3
billboard	0.29	0.61	0.39	0.27	5.4	6.2
lamppost	0.94	0.94	0.94	0.89	3.6	3.6
tree	0.88	0.81	0.84	0.99	6.5	7.2
Mean	0.76	0.81	0.77	0.82	4.8	4.2
Mean w/o bd.	0.85	0.85	0.85	0.93	4.7	3.8

TABLE III: **Reported results** of other pole recognition methods in geodesy, evaluated on different HDPC datasets. If multiple results are reported for multiple scenes, we manually averaged such results. Parameter errors E have not been estimated by listed works.

Class	Method	RE	PR	F_1	ACC
tree	Rutzinger *et al.* [7]	0.88	0.92	0.89	-
lamppost	Zheng *et al.* [6]	0.93	0.93	0.93	-
mixed	Ordóñez *et al.* [8]	0.91	-	-	0.96

VI. Conclusions

In this paper, we propose a novel deep learning-based method for detecting and classifying pole-like objects in high-density point clouds. To this end, we modify a state-of-the-art deep neural network that leverages learned encodings for the LiDAR point cloud by incorporating a second encoder stage and a feature pyramid. We consider two data representations of the point cloud as input to the neural network. While a sparse representation without ground points improves network speed, and may be considered for onboard usage, the best performance is achieved using a dense representation including ground points with all network modifications. Our method achieves a mean recall, precision, and classification accuracy of 0.85, 0.85, and 0.93, respectively. Thus, the approach is also suitable for geodesy applications such as a semi-automatic generation of high-definition maps.

References

[1] D. Wilbers, C. Merfels, and C. Stachniss, "Localization With Sliding Window Factor Graphs on Third-Party Maps for Automated Driving," in *Proc. of ICRA*, Montreal, Canada, May 2019, pp. 5951–5957.

[2] R. Spangenberg, D. Goehring, and R. Rojas, "Pole-Based Localization for Autonomous Vehicles in Urban Scenarios," in *Proc. of IROS*, Daejeon, Korea, Oct. 2016, pp. 2161–2166.

[3] A. Schaefer, D. Büscher, J. Vertens, L. Luft, and W. Burgard, "Long-Term Urban Vehicle Localization Using Pole Landmarks Extracted from 3-D Lidar Scans," in *Proc. of EMCR*, Prague, Czech Republic, Sep. 2019, pp. 1–7.

[4] C. Plachetka, N. Maier, J. Fricke, J.-A. Termöhlen, and T. Fingscheidt, "Terminology and Analysis of Map Deviations in Urban Domains: Towards Dependability for HD Maps in Automated Vehicles," in *Proc. of IV - Workshops*, virtual, Oct. 2020, pp. 63–70.

[5] Y. Yu, J. Li, H. Guan, C. Wang, and J. Yu, "Semiautomated Extraction of Street Light Poles From Mobile LiDAR Point-Clouds," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 3, pp. 1374–1386, 2015.

[6] H. Zheng, R. Wang, and S. Xu, "Recognizing Street Lighting Poles From Mobile LiDAR Data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 1, pp. 407–420, 2017.

[7] M. Rutzinger, A. Pratihast, S. Oude Elberink, and G. Vosselman, "Detection and Modelling of 3D Trees from Mobile Laser Scanning Data," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 38, no. 5, pp. 520–525, 2010.

[8] C. Ordóñez, C. Cabo, and E. Sanz-Ablanedo, "Automatic Detection and Classification of Pole-Like Objects for Urban Cartography Using Mobile Laser Scanning Data," *Sensors*, vol. 17, no. 7, 2017.

[9] Y. Li, W. Wang, X. Li, L. Xie, Y. Wang, R. Guo, W. Xiu, and S. Tang, "Pole-Like Street Furniture Segmentation and Classification in Mobile LiDAR Data by Integrating Multiple Shape-Descriptor Constraints," *Remote Sensing*, vol. 11, no. 24, 2019.

[10] J. Tu, J. Yao, L. Li, W. Zhao, and B. Xiang, "Extraction of Street Pole-Like Objects Based on Plane Filtering From Mobile LiDAR Data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 1, pp. 749–768, 2021.

[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 *Proc. of CVPR*, Long Beach, CAL USA, Jun. 2019, pp. 12 689–12 697.

[12] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proc. of CVPR*, Salt Lake City, UT, USA, Jun. 2018, pp. 4490–4499.

[13] L. Li, Y. Li, and D. Li, "A Method Based on an Adaptive Radius Cylinder Model for Detecting Pole-Like Objects in Mobile Laser Scanning Data," *Remote Sensing Letters*, vol. 7, no. 3, pp. 249–258, 2016.

[14] Z. Kang, J. Yang, R. Zhong, Y. Wu, Z. Shi, and R. Lindenbergh, "Voxel-Based Extraction and Classification of 3-D Pole-Like Objects From Mobile LiDAR Point Cloud Data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 11, pp. 4287–4298, 2018.

[15] Z. Shi, Z. Kang, Y. Lin, Y. Liu, and W. Chen, "Automatic Recognition of Pole-Like Objects from Mobile Laser Scanning Point Clouds," *Remote Sensing*, vol. 10, no. 12, 2018.

[16] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-Based Recognition of 3D Point Clouds in Urban Environments," in *Proc. of ICCV*, Kyoto, Japan, Sep. 2009, pp. 2154–2161.

[17] J. Huang and S. You, "Pole-Like Object Detection and Classification from Urban Point Clouds," in *Proc. of ICRA*, Seattle, WA, USA, May 2015, pp. 3032–3038.

[18] ——, "Point Cloud Labeling using 3D Convolutional Neural Network," in *Proc. of ICPR*, Cancun, Mexico, Dec. 2016, pp. 2670–2675.

[19] B. Li, "3D Fully Convolutional Network for Vehicle Detection in Point Cloud," in *Proc. of IROS*, Vancouver, Canada, Sep. 2017, pp. 1513–1518.

[20] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks," in *Proc. of ICRA*, Marina Bay Sands, Singapore, Jun. 2017, pp. 1355–1361.

[21] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proc. of CVPR*, Honolulu, HI, USA, Jul. 2017, pp. 77–85.

[22] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, pp. 3337–3354, 2018.

[23] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-Based 3D Single Stage Object Detector," in *Proc. of CVPR*, virtual, Jun. 2020, pp. 11 037–11 045.

[24] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "STD: Sparse-to-Dense 3D Object Detector for Point Cloud," in *Proc. of ICCV*, Seoul, Korea, Oct. 2019, pp. 1951–1960.

[25] P. Hu and D. Ramanan, "Finding Tiny Faces," in *Proc. of CVPR*, Honolulu, HI, USA, Jul. 2017, pp. 1522–1530.

[26] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proc. of CVPR*, Honolulu, HI, USA, Jul. 2017, pp. 936–944.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, "SSD: Single Shot Multibox Detector," in *Proc. of ECCV*, Amsterdam, The Netherlands, Oct. 2016, pp. 21–37.

[28] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[29] P. J. Huber, "Robust Estimation of a Location Parameter," *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

View publication stats