PEDESTRIAN DETECTION AND TRACKING IN SPARSE MLS POINT CLOUDS USING A NEURAL NETWORK AND VOTING-BASED APPROACH

Björn Borgmann[a,b]*, Marcus Hebel[a], Michael Arens[a], Uwe Stilla[b]

[a] Fraunhofer IOSB, Ettlingen, Fraunhofer Institute of Optronics, System Technologies and Image Exploitation.
Fraunhofer Center for Machine Learning. Gutleuthausstr. 1, 76275 Ettlingen, Germany
(bjoern.borgmann, marcus.hebel, michael.arens)@iosb.fraunhofer.de
[b] Photogrammetry and Remote Sensing, Technische Universitaet Muenchen. Arcisstr. 21, 80333 Munich, Germany
stilla@tum.de

KEY WORDS: Mobile laser scanning, LiDAR, pedestrian detection, object detection, tracking, neural network

ABSTRACT:

This paper presents and extends an approach for the detection of pedestrians in unstructured point clouds resulting from single MLS (mobile laser scanning) scans. The approach is based on a neural network and a subsequent voting process. The neural network processes point clouds subdivided into local point neighborhoods. The member points of these neighborhoods are directly processed by the network, hence a conversion in a structured representation of the data is not needed. The network also uses meta information of the neighborhoods themselves to improve the results, like their distance to the ground plane. It decides if the neighborhood is part of an object of interest and estimates the center of said object. This information is then used in a voting process. By searching for maxima in the voting space, the discrimination between an actual object and incorrectly classified neighborhoods is made. Since a single labeled object can be subdivided into multiple local neighborhoods, we are able to train the neural network with comparatively low amounts of labeled data. Considerations are made to deal with the varying and sparse point density that is typical for single MLS scans. We supplement the detection with a 3D tracking which, although straightforward, allows us to deal with objects which are occluded for short periods of time to improve the quality of the results. Overall, our approach performs reasonably well for the detection and tracking of pedestrians in single MLS scans as long as the local point density is not too low. Given the LiDAR sensor we used, this is the case up to distances of $22\,\mathrm{m}$.

1. INTRODUCTION

The detection and tracking of pedestrians and, in general, objects of certain types of interest like road users or road side objects is an important capability for several use cases, especially in the context of driver assistance systems and autonomous driving. For such applications, the detection and differentiation of relevant object types allows to take special care for the behavior of certain road users and their safety. For example, a pedestrian can change his or her movement vector faster than a car. In addition, pedestrians and bicyclists are more vulnerable than cars. Therefore, it may be important for a vehicle in traffic to keep a greater safety distance from pedestrians than from other cars. A mobile sensor system which is able to detect and track pedestrians and cyclists can also be used to determine their movement patterns in an urban environment, which could be useful for urban traffic planning purposes.

Multiple kinds of established sensors are typically used for autonomous driving, for driver assistance systems, and for mobile sensor systems in general. This includes radar, cameras for visible and infrared light, ultrasonic sensors and LiDAR sensors. LiDAR sensors are able to provide accurate three-dimensional geometric information of the surroundings up to distances of several hundred meters. They are independent of external light sources and often have a comparatively wide field of view. For example, rotating laser scanners reach full $360°$ perpendicular to their rotation axis. They are an advantageous type of sensors for the detection and tracking of objects in the vehicle's entire vicinity. One of their disadvantages is their low data density compared to most cameras: e.g., a 3D point cloud resulting from a single $360°$ scan by a commercially available LiDAR sensor typically contains approximately 130.000 points[1]. In comparison, a single video frame from a full-HD camera contains more than 2.000.000 pixels. In many use cases, for example in the area of mobile mapping, this is offset by the possibility to accumulate multiple LiDAR scans into one large combined 3D point cloud of data recorded over a longer period of time. But such a data accumulation over multiple scans does not work well for moving objects in the recorded area and can therefore not be used for the detection and tracking of such objects. The local data density provided by a LiDAR sensor also varies depending on the distance between the captured scene and the sensor. Hence, the processing of MLS (mobile laser scanning) point clouds for the purpose of detecting and tracking of mobile objects has to deal with sparse data and a varying data density.

This paper presents and extents a machine learning approach to detect persons or other objects of interest in point clouds of single MLS scans, e.g., single $360°$ scans of a rotating laser scanner. Starting from the point cloud representing such a single scan, our approach generates local point neighborhoods, each with a well defined coordinate frame. These are then processed by a neural network to decide if the neighborhood is part of an object of interest and where it is located in relation to the center of that object. This information is used in a voting process which accumulates the results of multiple processed local point neighborhoods. Since an object instance provides data for multiple local neighborhoods, we are able to generate multiple

* Corresponding author

[1] Velodyne HDL-64E with ten rotations per second

training data examples from a single labeled object. This allows us to work with a smaller amount of hand-labeled data during the training phase. The main focus and contribution of this paper is the integration of certain meta information about the local point neighborhoods as additional input for the neural network, the consideration of the varying data density, and supplementing the detection method with a basic tracking component to track detected objects in a sequence of point clouds.

2. RELATED WORK

This section is divided into two parts. In the first part, we position our work in relation to studies related to the area of object detection based on LiDAR data, covering classical approaches relying on handcrafted features as well as approaches with features learned by a deep neural network. The second part refers to work in the field of neural networks and their use for the processing of point clouds.

2.1 Object detection based on LiDAR data

The task of detecting objects of certain classes of interest is often divided into two subtasks. The first one is to extract regions from the processed data which potentially contain an object of interest (hypothesis generation). For example, a segmentation into contiguous regions can be applied to achieve this. Such a segmentation can be performed by methods like *region growing* (Velizhev et al., 2012) or *DBSCAN* (Asvadi et al., 2017). A problem with segmentation approaches is their proneness to over- or undersegmentation, which has to be dealt with in the further processing. More recent approaches use a *Region Proposal Network (RPN)* instead (Zhou, Tuzel, 2017).

After object candidates have been extracted, typically a classifier is used to decide for each cluster if it is an object of interest and to determine its object class (hypothesis validation). Such a classifier can either rely on handcrafted features, like *support vector machines* (Navarro-Serment et al., 2010), *bag-of-words* (Behley et al., 2013) or the *random forest* (Fukano, Masuda, 2015) classifier. Recently, many approaches use deep neural networks which learn both the features and the classification of these features. Hence, they do not rely on handcrafted features and are able to learn features which are better suited for the task at hand. The use of neural networks for the processing of point clouds is the focus of Subsection 2.2.

Voting-based approaches are able to detect objects in data without the need of an explicit hypothesis generation. Such methods extract features, which are then used to fill a voting space with votes for objects of interest. Objects are detected by searching for groups of matching votes in this voting space. Voting-based approaches can be implemented with handcrafted features and a dictionary to which the extracted features are matched. This dictionary is the result of a previous training phase and it is used for the casting of votes. (Velizhev et al., 2012) use such an approach on point clouds, but combine it with a preceding segmentation. (Knopp et al., 2011) use a voting-based approach to detect objects in a mesh of a 3D scene. This mesh has previously been generated from a 3D point cloud. Recently, (Qi et al., 2019) have proposed *VoteNet*, which processes a point cloud and uses a deep neural network for the casting of votes and to propose objects and classify them based on these votes.

In our earlier work we combined a neural network with a classical voting-based approach (Borgmann et al., 2019), and we follow the same strategy in the present paper. We use the network to cast votes and to replace the dictionary found in classical voting-based approaches (e.g., ISM). But in contrast to (Qi et al., 2019), we apply classical methods for the evaluation of the resulting voting space and process only local point neighborhoods with the neural network. This allows us to use smaller neural networks that require less training data.

2.2 Neural networks for point cloud processing

Several approaches for the use of neural networks on point clouds can be found in literature. One difficulty is the often unstructured nature of measured 3D point clouds. In contrast to camera images which have a defined pixel structure (regular grid), point clouds of many common LiDAR sensors do not have such an inherent structure. Neural networks which rely on discrete convolutions, very prominently used in the area of image exploitation, can therefore not directly be transferred to the processing of 3D point cloud data. A way to deal with this problem is the discretization of the point clouds. One possibility is the discretization into two-dimensional depth images or, if there is a source for color information (for example a camera which covers the same area as the LiDAR-sensor), in RGB depth images. Such a method is used by (Asvadi et al., 2017) to process MLS data for the detection of vehicles. The first steps of their approach are a ground removal and a segmentation based on DBSCAN. After that, the segments are converted into dense depth images which are then processed in a convolutional neural network to determine their object classes. An example for the use of RGB depth images and deep convolutional neural networks for the classification of objects has been presented in (Socher et al., 2012).

Instead of two-dimensional depth images, point clouds can also be converted to voxel grids. Voxels allow the processing by neural networks using discrete 3D convolutions. (Maturana, Scherer, 2015) describe an approach which detects objects in a 3D occupancy grid. They generate these grids from LiDAR and RGB depth point clouds. Similarly, (Garcia-Garcia et al., 2016) use occupancy grids and a convolutional neural network for object recognition tasks. (Zhou, Tuzel, 2017) detect cars and other road users. They use a voxel grid and determine a feature for each non-empty voxel. For the generation of this voxel feature, a neural network is used that generates a feature for each point, based on its coordinates and its coordinates in relation to the local mean of all points of the voxel. Following that, a max-pooling is used to combine the point features of all points of a voxel to a feature of the voxel itself. These voxel features are then processed in a convolutional neural network.

The conversion of unordered point clouds to a pixel or voxel structure is often difficult, since an optimal voxel or pixel size, to some degree, depends on the data density. If the size is too low for the available data density, many pixels or voxels turn out to be empty. If it is too high, a large amount of the information provided by the point clouds can get lost, because many points end up being inside the same voxel or pixel. This problem can not easily be solved, since the density provided by a LiDAR sensor depends on the distance between the sensor and the recorded area. Hence, it highly varies throughout a point cloud of a single scan. For this reason, it can be beneficial to directly process data of unordered point clouds in deep neural networks.

PointNet presented by (Qi et al., 2017a) is such a neural network that is able to process unordered sets of 3D points. The network learns a symmetric function to generate a feature describing the processed data. This can then be further processed for the classification or semantic labeling of the processed data. The network contains a special subnetwork which predicts affine transformations to deal with uncertainties with regard to the position and orientation of the processed data in the surrounding coordinate frame. The original PointNet was later extended to *PointNet++* by adding a hierarchical component (Qi et al., 2017b). At first, features for local subsets (each defined by a centroid point) of a point cloud are generated using the original PointNet. Neighboring subsets and their features are then combined across multiple hierarchical levels. This resembles the idea of convolutional neural networks of generating higher level features of a larger area from lower level features of a smaller area.

(Liu et al., 2019) presented an continuous convolutional neural network for point clouds which does not depend on discrete data. They use a specially designed convolution operator *PConv* which is able to deal with the irregularities of point clouds. They also connect multiple convolutional layers in a dense way using the output of all previous layers as input of the subsequent layer.

3. PROPOSED APPROACH

In the following we present our approach for the detection of pedestrians or, in general, distinct objects in sparse 3D point clouds. In addition, we present a basic tracking method for detected objects. We assume that the input is a sequence of single LiDAR scans (e.g., single 360° rotations of a LiDAR sensor with a rotating scanner head). The LiDAR data are expected to be directly georeferenced, i.e., resulting in 3D point clouds which share a common coordinate frame. This requires the availability of a technical (IMU/GNSS) or procedural (SLAM) way to take the movement of the sensor platform into account, which is not further discussed in this paper. We also assume that one axis of the coordinate frame is aligned with the gravitational axis (height). Within this paper, we call this axis the z-axis. Additionally we assume that, while recording the data, the position or trajectory of the LiDAR sensor is known in a way which allows us to store a viewpoint for each recorded single scan or even each recorded 3D point.

The basic outlines of our approach have already been described in (Borgmann et al., 2019). This paper focuses on improvements of the original approach: integration of certain meta information as additional input for the neural network, methodical refinements to better deal with the effects caused by varying data density, and the integration of a tracking component.

Our approach uses a neural network which, similar to the one presented by (Qi et al., 2017a), directly processes 3D points. In a preprocessing step and as input for the network, we divide the point cloud into local point neighborhoods. The output of the neural network is assessed in a voting process which is inspired by *implicit shape models* (Velizhev et al., 2012).

At first, we give an overview and describe the main processing steps. Then we explain the method used for the estimation of the ground level which, among other things, is a source for the newly added meta information, which acts as an additional input to the neural network. The neural network itself and the specifications of the meta information are topics of the sections 3.3 and 3.4. Following that, we explain the training of the neural network and finish with a short description of the tracking method used to deal with detected objects which are occluded for a short period of time.

3.1 Overview

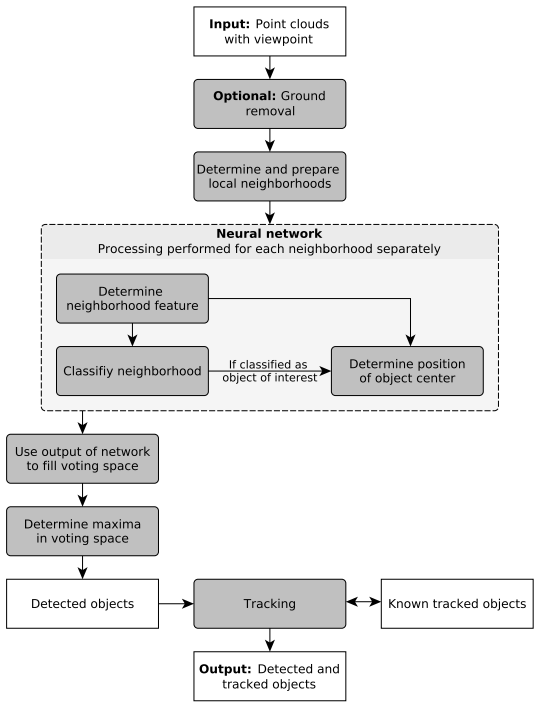

Input: Point clouds with viewpoint
Optional: Ground removal
Determine and prepare local neighborhoods
Neural network
Processing performed for each neighborhood separately
Determine neighborhood feature
Classifiy neighborhood
If classified as object of interest
Determine position of object center
Use output of network to fill voting space
Determine maxima in voting space
Detected objects
Tracking
Known tracked objects
Output: Detected and tracked objects

Figure 1. Main processing steps of our approach. The ground removal is optional and improves the runtime performance.

Figure 1 shows the processing steps of the extended approach presented in this paper. The initial ground removal is optional and allows for a faster processing by excluding the ground points from the further processing. The ground level estimation used for the ground removal is explained in Section 3.2.

After the ground removal we generate local point neighborhoods. These neighborhoods can be generated for each point of the processed data, but can also be generated only for a randomly selected subset of that data. This depends on a chosen sub-sampling parameter. In our previous work (Borgmann et al., 2019) we found that a low sub-sampling rate has only a minor influence on the quality of the results, but it improves the runtime performance significantly. Hence, for the purposes of this paper, we use a moderate sub-sampling rate by generating local point neighborhoods for approximately one third of the data.

Each local point neighborhood is defined by a center point and contains all surrounding points within a certain radius. This radius has to be chosen according to the type of processed data and the use case. The neighborhoods should provide some sort of pre-segmentation of the data, which means that the radius should be chosen in a way that usually only points of one or maybe a few different objects are part of the same neighborhood. We have analyzed the effects of the neighborhood radius in our earlier work. For the purposes of this paper, we use $0.5\,\mathrm{m}$ as neighborhood radius.

A local point neighborhood has a well defined coordinate frame which uses the center point as origin and is aligned in a way that the z-axis points upwards. The x-axis is perpendicular to the z-axis. It is also aligned with the line between the central point of the neighborhood and the viewpoint, pointing away from this point. The y-axis is defined by the other two axes following the rules for a right-handed coordinate frame.

The local point neighborhoods are processed by the neural network (see Section 3.3). This network determines the object class of each processed neighborhood. If a neighborhood is classified with a high certainty as part of an object of interest, an additional regression subnetwork estimates the position of that object's center in the coordinate frame of the neighborhood.

The output of the neural network is used for the generation of votes. Such a vote has three attributes:

1. Class of the object for which the vote is being cast
2. Center position of the object
3. Weight of the vote

The first attribute is set using the classification result of the neural network. The center position of the object is set same as estimated by the neural network, but transformed from the (local) neighborhood's coordinate frame back to the (global) point cloud's coordinate frame. The weight is determined using the following formula:

$$W_c = \frac{P(c)}{n} \tag{1}$$

where W_c = Weight of object candidate of class c
$P(c)$ = Probability of or confidence for class c
n = Amount of points in neighborhood radius

The parameter n accounts for the local point density and approximates the number of neighborhoods and votes generated in the local vicinity. Hence, it balances the effect that more votes are generated in areas with a higher data density.

The last step of the proposed object detection method is the search for maxima in the voting space. To achieve this, we consider the votes as object candidates and use their weight as score of these candidates. This score gets re-evaluated, taking into account the weight of neighboring candidates for an object of the same class. Based on the proximity, a portion of the score of such neighboring candidates is added to each candidate's score. After that, a threshold is applied removing all candidates with a score that is too low. Remaining candidates for the same type of object in close proximity to each other get merged. The remaining candidates are the final output of the object detection method. To generate bounding boxes for the detected objects, we consider the neighborhoods whose votes have contributed to the detection of an object. Finally, the bounding box of each object includes the central points of all its contributing neighborhoods.

3.2 Estimation of the ground level

The estimation of the ground level and the subsequent distinction between ground points and non-ground points is based on a two-dimensional grid which stores the height of the ground for each grid cell. The process is divided into three steps:

1. Initialization of the ground grid.
2. Validation of each grid cell using region growing based on a maximum steepness threshold.
3. Determine the distance of each point to a plane defined by the height values of the three nearest grid cells.

For the initialization of the ground grid, all points of the processed point cloud are assigned to grid cells according to their x- and y-coordinate. Then an initial height value is determined for each cell. This is achieved by sorting all points according to their z-coordinate. To deal with outliers, we use the z-coordinate of the point at the 0.05 quantile as height value of the cell.

The following validation step is needed since we cannot assume that every grid cell actually contains ground points. I.e., we intend to remove cells from the grid which do not include the ground level. We achieve this by picking a start cell for which we are reasonably sure that it includes ground points and then traverse the grid by region growing. The idea is that every valid ground cell should be reachable from the start cell without violating a criterion for the maximum steepness of the ground. Cells which cannot be reached are removed from the grid.

3.3 Topology of the neural network

We use a neural network which, similar to the PointNet approach (Qi et al., 2017a), directly processes the 3D coordinates of the member points in the local point neighborhoods. Figure 2 shows the structure of the proposed neural network. The first part of the network determines a descriptive neighborhood feature. This feature is mainly the result of the processing of the 3D coordinates in a multi-layer perceptron which uses shared layers. These shared layers lead to an invariance of the network with regard to the order of points in the input layer, which is important due to the unstructured nature of the processed point cloud. The neighborhood feature can be extended with certain meta information which are added to provide some basic context information to the processing of the local point neighborhood by the following subnetworks (see also Section 3.4).

The neighborhood feature is used in a classification subnetwork to determine the type of object the neighborhood is likely to be part of. This classification network uses three fully connected layers. There is an additional subnetwork to estimate the object position for object types of interest. One individual instance of this regression subnetwork is used for each object type of interest. It outputs a 3D coordinate in the coordinate frame of the local point neighborhood.

The network uses batch normalization after each layer with the exception of the output layers and the layers directly before these output layers. Following the recommendations of (Li et al., 2019) for the combined usage of batch normalization and dropout, we only use one dropout layer for each output which is directly after the last batch normalization layer of that output.

3.4 Integration of additional meta information

Our concept of processing local subsets of the data (local point neighborhoods) in the neural network has the advantage that the network does not have to learn much context information about objects of interest. We try to detect such objects based on their appearance and not based on their surroundings. This leads to moderate needs for the amount of labeled data during the training phase, as we have shown in our earlier work. However, some context information turned out to be quite beneficial to improve the detection results. Therefore, we decided to add at least some context information to the neighborhood features

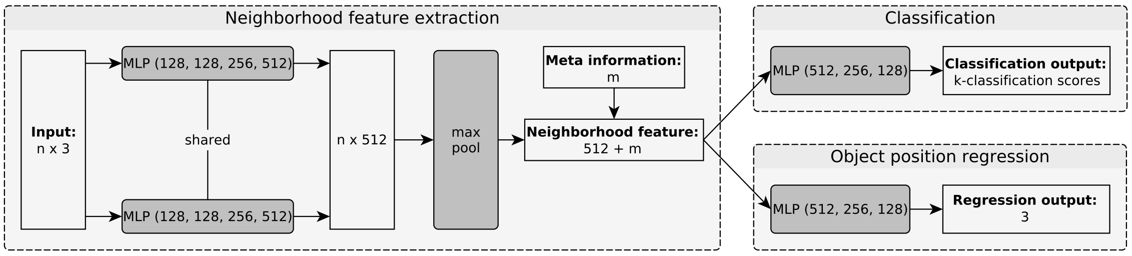


Figure 2. The proposed topology of the neural network. The main input are the n 3D points of a local point neighborhood. m meta information values are a supplemental input. Outputs are classification scores for k classes and estimated 3D coordinates of the object's center. Batch normalization is used for all MLP (*muli-layer perceptron*) layers, except for the layer directly before the output layer. The last batch normalization of each output is followed by a dropout layer with a dropout rate of 0.2.

processed in the classification and regression subnetworks. In the context of this paper, we tested two types of such additional meta information, which are both easy to determine.

The first value to supplement the neighborhood feature is the distance between the LiDAR sensor and the processed local point neighborhood (at its center point). The idea is that the appearance of an object in the data can differ depending on the local data density. This density, in turn, depends on the distance between object and LiDAR sensor. Hence, providing this distance to the classification and regression subnetworks could be beneficial.

The second value we integrate is the height above ground of the processed neighborhood. The idea is that certain local features of an object typically occur only at certain heights on that object. For example, the head is usually located higher than the feet of a pedestrian. In addition, the objects we are mainly interested in are typically standing or moving on the ground level, e.g., for traffic safety it is not necessary to detect persons on balconies. To determine the height above ground, we use the estimation of the ground level described in Section 3.2 and determine the distance between the ground grid and the processed neighborhood. For local point neighborhoods in areas which are not covered by the ground grid, we use the average ground height of the grid as fallback value.

3.5 Training phase of the neural network

The training phase of the neural network is divided into two steps. At first, we train the feature extraction and classification parts of the network. After that, we train one instance of the regression part for each object type of interest using the feature extraction part which we have trained in the first step without modifying it further. For the actual training we apply an *Adam* optimizer (Kingma, Ba, 2014) with a learning rate of 0.0004. To prevent overfitting, we validate the training progress with separate validation data and stop each training step if no further improvement can be made for five consecutive training epochs. As the result of the training phase we keep the weights of the best performing epoch.

The training data that we use are manually labeled point clouds of single scans. For each object instance labeled in these point clouds, the object position, a bounding box, and all points which are considered as part of the object are known. Since we process local point neighborhoods in the neural network, we can generate a multitude of training examples from a single labeled object. One local point neighborhood can be generated for each point of an labeled object and used for the training. This allows us to reasonably work with a moderate number of labeled 3D objects for the training. In addition, we generate negative examples for the training by randomly selecting points in the data to generate local point neighborhoods which are not part of any labeled object.

Due to the varying point density of MLS point clouds, generating a training sample for each possible neighborhood of a labeled object carries the risk that object instances in close proximity to the sensor are overrepresented due to the higher point density, whereas object instances in greater distances are underrepresented in the training samples. This effect could lead to a neural network whose detection and classification performance is comparatively low in areas with a low point density. To compensate for this effect, we increase the weight of training samples in greater distances, giving them a higher impact on the training than samples recorded in close proximity to the LiDAR sensor.

3.6 Tracking of detected objects

We added a basic tracking component to our approach to be able to deal with temporarily occluded objects and to better determine the movement (speed and direction) of detected objects. The tracking is based on a Kalman filter with a constant velocity model. Such a tracker is easy to implement to work with our detection method, since the resulting 3D object coordinates are given in a global coordinate frame and can directly be used for the tracking. A constant velocity model seems to be a good fit for pedestrians who can rapidly change their movement vector. A constant acceleration model may be better suited for vehicles like cars.

The matching between tracked objects and detected objects is done by simply comparing their geometric position. This is a baseline method and, in future work, we plan to supplement it with a comparison of the features of the local point neighborhoods.

If a tracked object is no longer detected, we keep track of it until the variance of the tracker for the object position is higher than a certain threshold. Since this variance describes how accurate the predicted position of a tracked but undetected object is, it is a good criterion to decide which objects cannot be tracked further with sufficient accuracy.

4. EXPERIMENTS

We conducted experiments with several goals: Determine the performance of the presented approach depending on the local

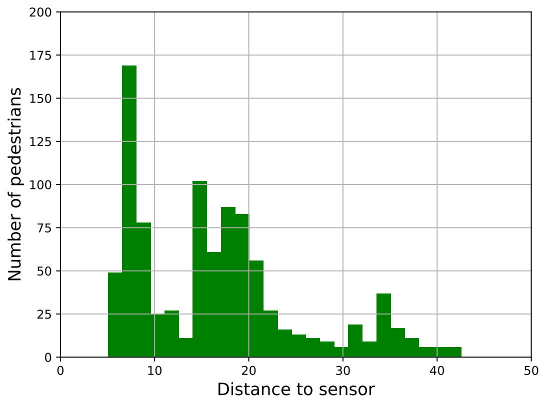


Figure 3. Measuring distances to fully visible pedestrians occurring in the evaluation data

point density, determine the impact of integrating additional meta information, and determine the performance and benefits of the tracking method. The experiments were conducted with the focus on the detection and tracking of pedestrians in an urban road environment.

For the experiments we used LiDAR data which we recorded using a multi-sensor vehicle. We have presented this vehicle in detail in an earlier work (Borgmann et al., 2018). In addition to being a multi-camera vehicle, the vehicle is equipped with several LiDAR sensors. However, only one of these sensors was configured to capture the entire 360° vicinity of the vehicle, and we only used data recorded by this sensor. It is a Velodyne HDL-64E that performs about 1.300.000 range measurements per second which are distributed over 64 scan lines. While recording the data used for our experiments, the rotating head of the sensor was rotating ten times per second, hence a single 360° scan contains about 130.000 measurements. Not all of these measurements result in meaningful 3D points, since some are directed into the sky, deliver no result due to low scene reflectance, or measure parts of the measurement vehicle itself. After filtering the measured points, an approximate average of 95.000 meaningful scene points remain in the point cloud of each single scan. The vehicle is also equipped with an inertial navigation system (INS) which we used for the direct georeferencing of the recorded data. As a result, all point clouds share the same ENU coordinate frame.

The data available for the experiments were divided into three groups: For the training of the neural network, multiple short sequences of labeled point clouds recorded in road traffic and a sequence of a staged scene were used. These training data consist of 1300 labeled point clouds in total. During the training phase, a second group of 226 additional point clouds were used to validate the training progress and to detect the occurrence of overfitting. The third data group was used for the actual evaluation: a longer sequence containing 300 labeled point clouds. This sequence was recorded at an intersection and contains pedestrians, bicyclists and other road users in different distances. The area of these evaluation recordings was neither covered by the training nor the validation data.

The sequence used for the evaluation contains 941 instances of pedestrians, additional 1105 instances of pedestrians with occlusions, 307 instances of cyclists and 790 instances of cyclists with occlusions. Unfortunately, cyclists are not part of the labeled data used for the training. Hence, we were unable

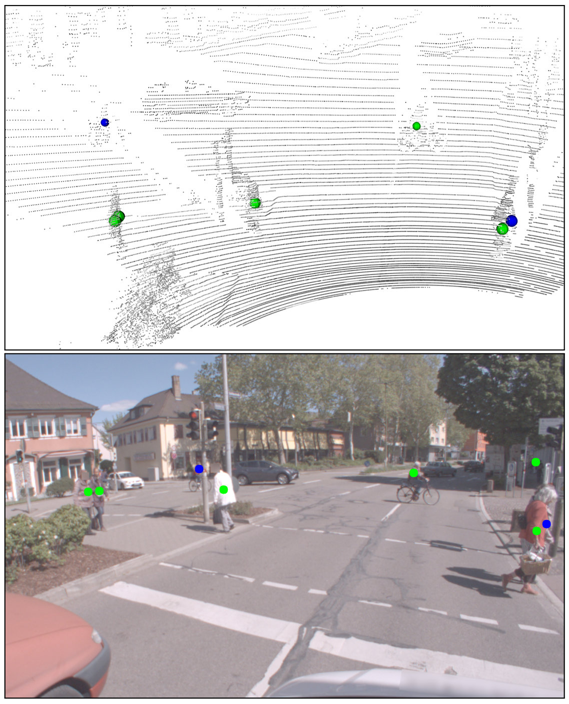


Figure 4. **Top:** Visualization of an exemplary output of our approach: detected persons are marked by green dots. Currently undetected but tracked persons are predicted by a blue dot. **Bottom:** The same scene shown in a synchronously recorded video, with all 3D markings projected to the image.

to train the neural network sufficiently to discriminate between pedestrians and cyclists. For the purpose of the evaluation, we made the decision to ignore cyclists completely, neither counting them as true positive, nor false positive, nor false negative results. For future work, we plan to generate additional labeled training data to close that gap. Figure 3 shows the distribution of the pedestrians in the evaluation data with regard to the measured distance.

The output of our approach are the positions of detected and tracked pedestrians as shown in the example at the top of Figure 4. These results were compared with the ground truth to determine the number of true positive (tp), false positive (fp), and false negative (fn) detections. These numbers were used to calculate *precision* and *recall*, which are defined as follows:

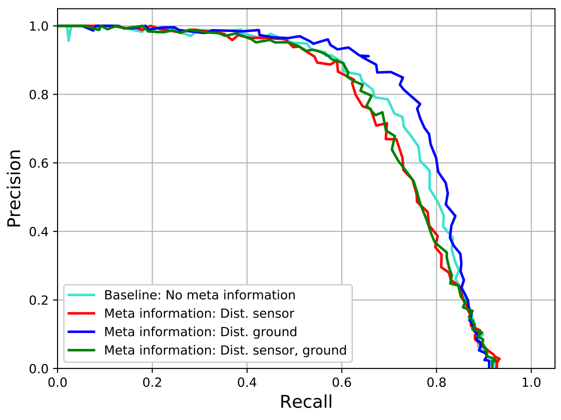
$$Precision = \frac{tp}{tp + fp} \tag{2}$$

$$Recall = \frac{tp}{tp + fn} \tag{3}$$

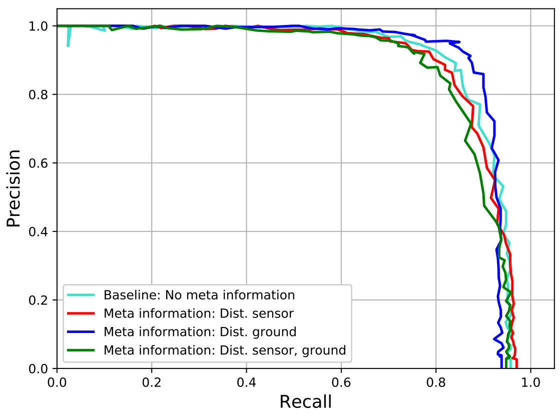
4.1 Impact of point density and integration of meta information

We only considered those pedestrians in the evaluation data which are not significantly occluded. The detection performance was evaluated under varying point density as well as concerning the influence of integrating meta information of local point neighborhoods. Besides that, we did not use the tracking component for this analysis. We compared four configurations:

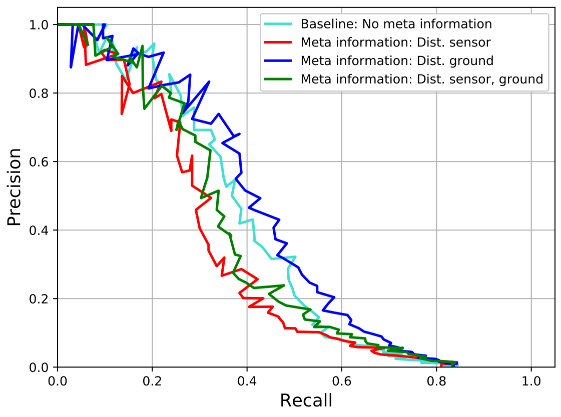
- Baseline: Without any consideration of meta information.



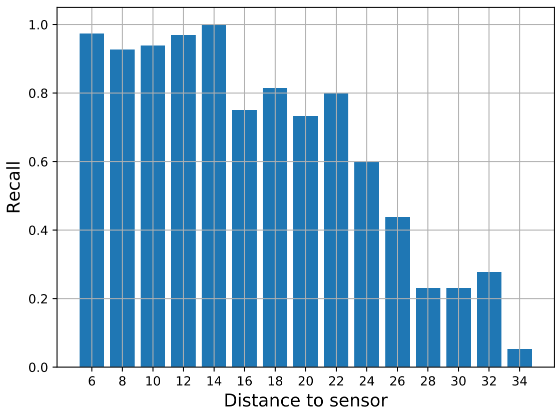
(a) Results for pedestrians in all distances



(b) Results for pedestrians in less than 20 m distance



(c) Results for pedestrians in more than 20 m distance



(d) Recall for different distances at an overall precision of 0.8 using the best performing configuration (considering meta information *distance to ground*, but not *distance to sensor*)

Figure 5. (a), (b), (c): Comparison of the four analyzed configurations for different distance intervals. (d) Analysis of the performance in different distances to the sensor.

- Using the meta information *distance to sensor*.
- Using the meta information *distance to ground*.
- Including both *distance to sensor* and *distance to ground*.

In addition, we took the distance between the recorded pedestrians and the LiDAR sensor as an indicator for a high or low point density, and we analyzed the influence of this parameter on the detection performance. Hence, we evaluated the capability to detect pedestrians in different distances.

The results of this evaluation are shown in Figure 5. With regard to integrating the meta information, we achieved mixed results: The *distance to ground* meta information is clearly beneficial and increases the detection performance at all distances, as shown in Figure 5a. To get a better impression, we split this evaluation considering only short distances (Figure 5b) and long distances (Figure 5c). The *distance to sensor* meta information, on the other hand, has an adverse effect on the detection method, performing worse than in the baseline configuration. This also shows up in the configuration which uses both *distance to sensor* and *distance to ground*. A possible explanation is that integrating the meta information *distance to sensor* causes the training of the network to no longer generalize well for objects in different distances. Hence, the training data, to a much higher degree, has to cover objects of interest in all relevant distances. This also means that a much larger amount of training data would be necessary. We consider these costs as too high for the potential improvements.

With regard to the detection performance in different distances (i.e., different point densities), Figure 5d shows that our approach performs very good up to distances of about 14 m and still reasonably well up to distances of about 22 m. In distances beyond that, the detection performance decreases rapidly. This is also shown when comparing the performance separately for the detection of pedestrians in the distance interval up to 20 m (Figure 5b) and beyond that (Figure 5c).

4.2 Benefit of including a tracking component

To analyze the benefit of the tracking component with respect to the detection of occluded objects, we compared the performance of the configuration using the *distance to ground* meta information with and without tracking. The result of this evaluation is shown in Figure 6. For the evaluation, we considered fully visible as well as partly occluded pedestrians labeled in the ground truth data. A difficulty is that the ground truth contains no labels for pedestrians which are completely occluded in that scan. Since such instances can only be predicted and not

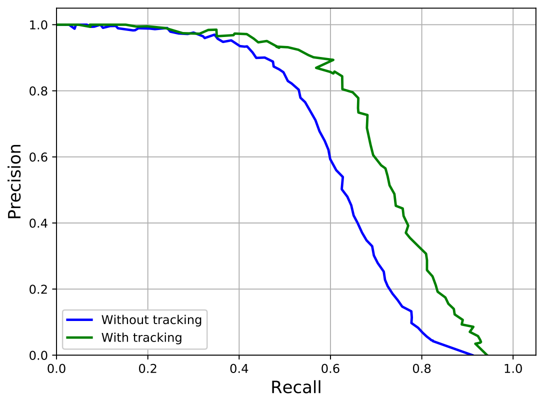


Figure 6. Comparison of results with and without tracking.

be detected, the clear advantage of including the tracking component would be even more apparent in this evaluation if these labels were available. We therefore conclude that even a basic tracking component is beneficial and clearly recommended.

5. CONCLUSION AND FUTURE WORK

We improved our existing approach for the detection of objects in point clouds of single MLS scans that have a varying and sparse point density. Our approach uses a neural network and a subsequent voting process to detect objects without the need for a preceding hypothesis generation step. The neural network processes the data subdivided into local point neighborhoods, of which the network uses the member points as input. In the context of this paper, we extended the input by providing the network with meta information on the processed neighborhoods to improve its performance. This has beneficial effects in case of one tested meta information (*distance to ground*), but adverse effects for another one (*distance to sensor*). We also included a basic tracking component, based on a Kalman filter, to our approach. This significantly increased the performance with regard to objects which are temporarily occluded. We compared the performance of our detection method in different distances, which correspond to different local point densities. For the sensor used in our specific experimental system, we achieved a satisfactory performance for the detection of pedestrians in distances up to $22\,\mathrm{m}$.

For future work, we plan to integrate additional sensors into the processing. We think that a combination of LiDAR-based detection and tracking with a camera-based local examination can provide useful additional information, e.g., about the road safety awareness of pedestrians. We also intend to improve the matching of tracked and detected objects in the proposed tracking component. Currently, we only consider the geometric proximity for the tracking, but it would for sure be beneficial to generate tracks by assigning the extracted features of local point neighborhoods.

REFERENCES

Asvadi, A., Garrote, L., Premebida, C., Peixoto, P., Nunes, U. J., 2017. DepthCN: Vehicle detection using 3D-LIDAR and ConvNet. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 1–6.

Behley, J., Steinhage, V., Cremers, A. B., 2013. Laser-based Segment Classification Using a Mixture of Bag-of-Words. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4195–4200.

Borgmann, B., Hebel, M., Arens, M., Stilla, U., 2019. Using neural networks to detect objects in MLS point clouds based on local point neighborhoods. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W7, 17–24.

Borgmann, B., Schatz, V., Kieritz, H., Scherer-Klöckling, C., Hebel, M., Arens, M., 2018. Data processing and recording using a versatile multi-sensor vehicle. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-1, 21–28.

Fukano, K., Masuda, H., 2015. Detection and Classification of Pole-like Objects from Mobile Mapping Data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5, 57–64.

Garcia-Garcia, A., Gomez-Donoso, F., Garcia-Rodriguez, J., Orts-Escolano, S., Cazorla, M., Azorin-Lopez, J., 2016. PointNet: A 3D Convolutional Neural Network for Real-Time Object Class Recognition. *2016 International Joint Conference on Neural Networks (IJCNN)*, 1578–1584.

Kingma, D. P., Ba, J., 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Knopp, J., Prasad, M., Gool, L. V., 2011. Scene Cut: Class-Specific Object Detection and Segmentation in 3D Scenes. *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, 180–187.

Li, X., Chen, S., Hu, X., Yang, J., 2019. Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2677–2685.

Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., Pan, C., 2019. DensePoint: Learning Densely Contextual Representation for Efficient Point Cloud Processing. *IEEE International Conference on Computer Vision (ICCV)*, 5239–5248.

Maturana, D., Scherer, S., 2015. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 922–928.

Navarro-Serment, L. E., Mertz, C., Hebert, M., 2010. Pedestrian Detection and Tracking Using Three-dimensional LADAR Data. *The International Journal of Robotics Research*, 29(12), 1516-1528.

Qi, C. R., Litany, O., He, K., Guibas, L. J., 2019. Deep Hough Voting for 3D Object Detection in Point Clouds. *Proceedings of the IEEE International Conference on Computer Vision.*

Qi, C. R., Su, H., Mo, K., Guibas, L. J., 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 77–85.

Qi, C. R., Yi, L., Su, H., Guibas, L. J., 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (eds), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 5099–5108.

Socher, R., Huval, B., Bath, B., Manning, C. D., Ng, A. Y., 2012. Convolutional-Recursive Deep Learning for 3D Object Classification. *Advances in Neural Information Processing Systems*, 656–664.

Velizhev, A., Shapovalov, R., Schindler, K., 2012. Implicit shape models for object detection in 3D point clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, I-3, 179–184.

Zhou, Y., Tuzel, O., 2017. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. https://arxiv.org/abs/1711.06396.