USING NEURAL NETWORKS TO DETECT OBJECTS IN MLS POINT CLOUDS BASED ON LOCAL POINT NEIGHBORHOODS

Björn Borgmann[1,2,*], Marcus Hebel[1], Michael Arens[1], Uwe Stilla[2]

[1] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB,
Gutleuthausstr. 1, 76275 Ettlingen, Germany
- (bjoern.borgmann, marcus.hebel, michael.arens)@iosb.fraunhofer.de
[2] Photogrammetry and Remote Sensing, Technical University of Munich (TUM), Germany - stilla@tum.de

KEY WORDS: Mobile laser scanning, LiDAR, feature learning, neural networks, object detection, pedestrians

ABSTRACT:

This paper presents an approach which uses a *PointNet*-like neural network to detect objects of certain types in MLS point clouds. In our case, it is used for the detection of pedestrians, but the approach can easily be adapted to other object classes. In the first step, we process local point neighborhoods with the neural network to determine a descriptive feature. This is then further processed to generate two outputs of the network. The first output classifies the neighborhood and determines if it is part of an object of interest. If this is the case, the second output determines where it is located in relation to the object center. This regression output allows us to use a voting process for the actual object detection. This processing step is inspired by approaches based on implicit shape models (ISM). It is able to deal with a certain amount of incorrectly classified neighborhoods, since it combines the results of multiple neighborhoods for the detection of an object. A benefit of our approach as compared to other machine learning methods is its low demand for training data. In our experiments, we achieved a promising detection performance even with less than 1000 training examples.

1. INTRODUCTION

The observation and perception of the surrounding environment is a major requirement of an autonomous car and many driver assistance systems. One task is the detection of potential obstacles or moving objects and, for instance, the utilization of that information to prevent collisions. For some use cases, the detection and classification of objects of distinct types is beneficial. For example, in order to control and safely operate an autonomous driving car, the behavior of pedestrians and other road users is of particular interest. Such information can be seen as an important input for risk assessment of the current driving situation. Only if an autonomous driving car is aware of object classes such as pedestrians and cyclists, it can keep an extra distance to these, taking into account that they are especially vulnerable and can change their moving direction more rapidly than most other road users.

The focus of this paper is on processing of LiDAR sensor data, since LiDAR is an established sensing technique to support and enable autonomous driving. LiDAR sensors are capable of directly gathering geometric information of the surrounding environment. For that reason, they are particularly useful to determine the position and moving direction of other objects in relation to the sensor system. In addition, they often have a large field of view (360° for rotating LiDAR scanners) and operate independent from natural or other external light sources. In comparison to cameras, LiDAR sensors typically have a lower resolution, which means that they provide a lower local data density. In another aspect, the data acquisition by moving and scanning LiDAR sensors (MLS) typically results in unstructured 3D data, whereas cameras produce 2D image data in a fixed matrix structure. In view of these different data structures, LiDAR sensor data is typically more difficult to handle than image data.

[*] Corresponding author

Common approaches for object detection and object recognition rely on handcrafted features. These features are extracted locally from the data and used by a classifier in order to determine if the local subset of data is part of an object of a certain type. For LiDAR data, which are most commonly represented as 3D point clouds, typical handcrafted features either use some general information of the 3D points themselves (e.g., the height above ground) or depend on relations to neighboring points. Generally speaking, such features describe the local neighborhood of a point. In some cases combinations of multiple features are used. In recent years, deep learning and neural networks have offered alternatives to overcome the dependence on handcrafted features. In contrast to classical approaches, handcrafted features are replaced by *learned* ones.

In this paper, on the one hand, we follow the classical approach of describing points using their local point neighborhood. On the other hand, we use features learned by a neural network with a *PointNet*-like architecture (Qi et al., 2017a) instead of handcrafted features. The neural network is then also used to perform parts of the object recognition itself by classifying the point neighborhoods and determining where they are located on a certain object in relation to the object center. These results of the neural network are used as a basis for a voting process to detect objects of a certain type in the processed data.

2. RELATED WORK

This section is divided into three parts. First we quote other work related to handcrafted point features in general. Then we summarize some related existing approaches for the detection of distinct objects in point clouds. The last part of this section gives a short overview on modern *deep learning* approaches for 3D point cloud data.

2.1 Local point features

There exist different types of handcrafted features for 3D points and 3D point clouds which are often designed for a specific task. We mention only a few of them that are relevant in the context of this paper. For example, Johnson and Hebert (1999) proposed *Spin Images* as a feature descriptor for object recognition in 3D data. Spin images are based on the surface normal of a point and on the *support* of the spin image. The support is defined as all neighbors in a certain distance whose estimated normal directions do not exceed a certain maximum angle to the normal vector of the point of interest. To determine a spin image, a cylindrical two-dimensional coordinate system is defined using the surface normal of the point of interest. The points of the support are grouped into bins and counted based on their coordinate in the cylindrical coordinate system.

Rusu et al. (2008) proposed *Point Feature Histograms* which describe the mean curvature at a point and are extracted using the point itself, its surface normal, its neighbors and the surface normals of the neighbors. They determine point pairs in the neighborhood and perform certain calculations for these pairs, which results in a high-dimensional histogram. The method was later improved (Rusu et al., 2009) to *Fast Point Feature Histograms*, which are computationally more efficient by removing some less effective components of the original point feature histograms.

Several features for a 3D point neighborhood can be extracted using the eigenvalues of the covariance matrix (principal component analysis). Weinmann et al. (2015) use such eigenvalues of a optimal neighborhood, which is carefully selected by an automatic method based on the processed data, to calculate local shape features like linearity, planarity, scattering, omnivariance, anisotropy, eigenentropy, and change of curvature.

The approach presented in this paper keeps the concept of extracting point features which are based on the local neighborhood of 3D points. Instead of using handcrafted features, we rely on features learned and extracted by a neural network.

2.2 Object detection and recognition using LiDAR data

Many approaches for object detection and recognition in 3D data first perform a segmentation of the data in order to separate potential objects from the background. Typically, clusters of points are found by a region growing procedure, which are then regarded as object candidates. After that, the candidates are evaluated by a classification method based on certain geometric features. For example, Navarro-Serment et al. (2010) use a *support vector machine* as classifier in such an approach. Support vector machines learn a hyperplane in feature space which differentiates between the different classes.

Behley et al. (2013) use a combination of several *bag-of-words* classifiers which differ in their parametrization of the point feature descriptors. They also use a hierarchical segmentation approach to deal with over- and under-segmentation problems. Bag-of-words classifiers use a dictionary of words which are associated to classes and described by a feature descriptor. To classify data, they search in the dictionary for words which best match the features of the data.

Implicit shape models (ISM) are a variant of bag-of-words approaches in the sense that the words are not only associated with a class but also with a relative position of the object of that class. If multiple matched words point at the same position and class, that is a strong indication for the actual presence of an object of that class near that position. Velizhev et al. (2012) use implicit shape models to detect objects of certain classes in 3D point clouds and use spin images as feature descriptor. In our earlier work we used ISM for the detection of pedestrians in segmented LiDAR-data (Borgmann et al., 2017) and later in unsegmented data of multiple LiDAR-sensors (Borgmann et al., 2018a). For the latter approach, we performed a data fusion in the voting space of the ISM. The approach presented in this paper extends that earlier work. We kept the voting process of the ISM, but replaced the extraction of handcrafted features and the dictionary of geometric words with a neural network.

2.3 Neural networks and LiDAR data

The recent success of *deep learning* with neural networks in the area of image processing led to different approaches to transfer this concept into the area of 3D point cloud processing. The main difficulty of such attempts lies in the unordered nature of point clouds. Images are organized in a fixed pixel-by-pixel matrix structure, which is ideal for processing by convolution operations in deep *convolutional neural networks*. Therefore, many attempts to transfer this concept to 3D data are based on converting the unorganized 3D data into a regular data structure. Socher et al. (2012) use a depth image representation instead of an irregular point cloud for the task of object classification. They are able to apply neural networks similar to the ones used for RGB images and actually include RGB information in their method. Although it is possible to generate depth images from most 3D sensor data, this process can be difficult depending on the involved type of sensor (e.g., a moving LiDAR scanner). This is particularly the case if LiDAR data of multiple sensors are fused. In addition, an interpolation and discretization of 3D data to depth image pixels often comes at the cost of an information loss.

Another ordered representation of 3D data is a voxel grid. Being the 3D equivalent of pixels, voxels can equivalently be used in convolutional neural networks for object detection and recognition tasks on 3D data (Maturana and Scherer, 2015; Garcia-Garcia et al., 2016). One difficulty when using voxels is the decision what size of voxel one should use. If the size is too small for the available data density, there is the problem of filling voxels for which there is no actual data available. If the size of the voxels is too large, information is lost. Since LiDAR sensors typically have a high data density in short distances and a lower one in larger distances, this problem becomes even more evident.

Qi et al. (2017a) proposed *PointNet*, an approach which directly processes point clouds in a neural network. The data are processed as an unordered list of points with their 3D coordinates. The network learns a symmetric function to generate a global feature for the complete processed cloud. Such a function does not depend on the order of the processed data. An additional sub-network is used to predict affine transformations to deal with the fact that the position and orientation of the processed data in the coordinate frame are uncertain but should not affect the results. The global feature is then further processed by additional layers of the network to generate a classification result for the whole processed cloud. Their network can also be used for semantic segmentation of the points of the cloud. To achieve this, the global feature is combined with a local feature, the output of an earlier layer of the network, and then processed by additional network layers to generate a per-point score for each semantic category.

Qi et al. (2017b) later added an hierarchical component to their original PointNet approach. The idea is to first generate local subsets of the processed data and to extract features for these subsets using PointNet. A subset is defined by a centroid point and its neighbors. Then, multiple of the original subsets and their features are combined to generate a new larger subset, which is once again processed by a PointNet network for feature extraction. This process repeats over multiple hierarchical levels. This resembles the idea of CNN networks to first extract local features and then group them together into higher level features. In comparison to the original PointNet, this hierarchical *PointNet++* approach is better able to deal with the non-uniform data density of typical point clouds recorded by LiDAR sensors.

The approach presented in this paper uses a neural network which, similar to the one presented by Qi et al. (2017a,b), processes an unordered list of points. Instead of using whole point clouds or well defined large subsets of point clouds, we consider individual points and their neighbors as input to generate results for the selected individual points. In a sense, this is similar to the lowest hierarchical level of the PointNet++ approach (Qi et al., 2017b). In the PointNet++ approach, neural networks are used to combine low-level features over multiple hierarchical levels, resulting in high-level features. Differing from this, we directly generate a classification result and an object localization for the points whose neighborhoods we process. These results are then further processed outside of a neural network in an ISM-like voting process to detect objects of interest in the processed data. Our approach is able to find all instances of objects types of interest in the processed data and does not rely on a prior segmentation of that data.

3. PROPOSED APPROACH

In the following we describe our approach for object recognition in point clouds. As input we assume 3D point clouds. We also assume that there is some kind of viewpoint information available (e.g., the sensor position while recording the data, the sensor's trajectory), either for the whole point cloud or for each 3D point individually. Finally, we assume that one axis of the coordinate frame is aligned with the direction of gravity or height. This axis is usually referred to as the z-axis.

The main processing steps of the approach proposed in this paper are shown in Figure 1. As in other approaches (Velizhev et al., 2012) and our earlier work (Borgmann et al., 2017, 2018a), we included a ground removal step which is not mandatory but can be beneficial with regard to the processing time. Typically, many data points acquired by an MLS system (i.e., a car equipped with LiDAR scanners) are captured at the ground level. These ground points are relevant to analyze the terrain navigability, but can be ignored during the detection of other road objects. Removing the ground points significantly reduces the amount of data to process without affecting the results of object detection.

After the optional ground removal, we determine and prepare local point neighborhoods. This step is described further in the following section. These neighborhoods are then processed by a neural network. The network provides two outputs: One classifies the neighborhood as part of an object of a certain type. The second output estimates the position of the center of that object. In this paper, we exemplary focus on the detection of pedestrians, but the approach can easily be adapted to other object classes. The topology of the neural network is described in Section 3.2.

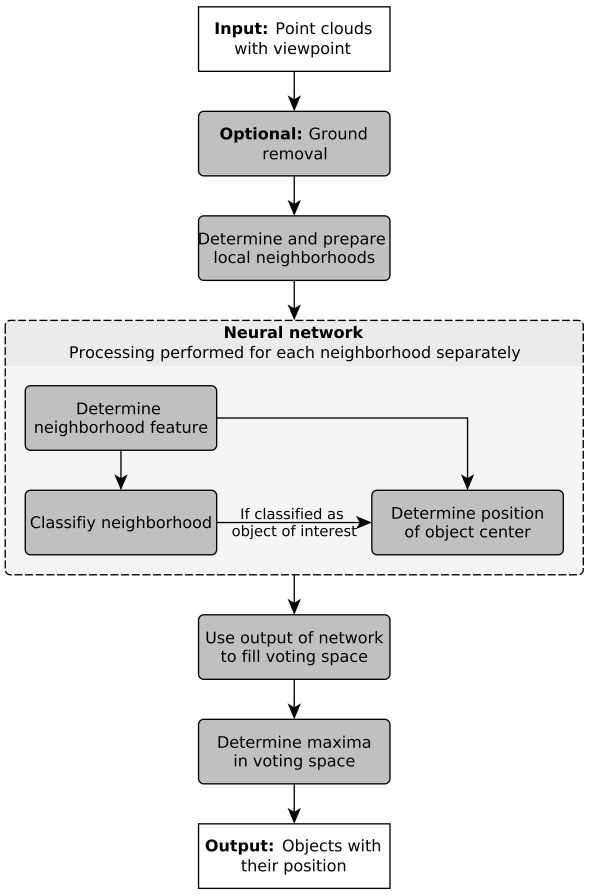

Input: Point clouds with viewpoint
Optional: Ground removal
Determine and prepare local neighborhoods
Neural network
Processing performed for each neighborhood separately
Determine neighborhood feature
Classifiy neighborhood
If classified as object of interest
Determine position of object center
Use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

Figure 1. Overview of the proposed approach

We use the output of the neural network to generate votes for objects at specific positions. These votes are comprised of a 3D coordinate for the object center and a weight. The idea is that multiple votes originating from parts of an actual object will accumulate a significant weight at roughly the same position. In case of local point neighborhoods for which the network delivers incorrect results, we can assume that their votes are randomly distributed in space and will not accumulate enough weight for one position to result in a false positive detection. This voting process is inspired by implicit shape models (ISM) and is designed to deal with a certain amount of wrong outputs of the neural network. More details of this particular processing step are described in Section 3.3.

3.1 Local point neighborhoods

We define the local neighborhood of a point as all its surrounding points in a certain fixed distance. For the approach proposed in this paper, a neighborhood defined by a fixed distance is beneficial as compared to k-nearest neighbors, since that would result in features which would not generalize well for different point densities. If there is only a small amount of points in the local point neighborhood, we do not process it further, since the information content of such a neighborhood is too low to produce useful results.

A local right-handed coordinate frame is defined for each of the processed point neighborhoods, which has its origin at the central point of the neighborhood. We orient the coordinate frame according to the direction of gravitation and the direction to the viewpoint. The z-axis of the local point neighborhood's coordinate frame is oriented upwards corresponding to the height axis. The x-axis is perpendicular to the z-axis and aligned with the line between the central point and the viewpoint, pointing away from the viewpoint. The y-axis is then defined to be perpendicular to the x- and z-axis with the direction that results in a right-handed coordinate frame. After that, the coordinates of all points of the local point neighborhood are transformed into this local coordinate frame.

Although we defined the local point neighborhood to have a fixed radius instead of k-nearest neighbors, we need a fixed input size for the neural network. A fixed amount of neighbors is achieved in two different ways: If we have less than the desired number of points in the neighborhood, we use padding and add the missing points which all have the coordinate frame's origin as their coordinates. If we have too many points in the neighborhood, we randomly select the needed amount and ignore the rest. We assume that randomly selecting a meaningful number of points sufficiently preserves the general shape of the neighborhood and that more sophisticated methods of selecting subsets of points are not required. This random selection of points effectively normalizes the point density of the local point neighborhood if it is above a maximum. This maximum is implicitly defined by the input size of the neural network.

Since local point neighborhoods with central points in close distance typically share many of their member points, they can end up being quite similar to each other. We expect that this fact would result in many redundancies in the following processing steps, unnecessarily reducing the run-time performance. Therefore we introduce an additional parameter s which allows us to generate and process only a subset of the possible neighborhoods. Although a more sophisticated selection of interest points could probably be beneficial, we stick to a simple sub-sampling method due to the difficult and computationally intensive nature of a selective interest point detection.

3.2 Design of the neural network

The chosen topology of the neural network is shown in Figure 2. It is inspired by the approach described by Qi et al. (2017a) and shares the main design ideas with their *PointNet* network. This includes the direct processing of the 3D coordinates of an unordered set of points. The network learns a symmetric function which is invariant to the order of the input vector. As compared to their layout, our network is simplified in several aspects, reducing its overall complexity.

Since we process local point neighborhoods instead of whole point clouds or point cloud segments containing whole objects, the number of points in the input vector is generally small. That means we can use a smaller amount of hidden layers with less neurons in each layer. In addition, the input data are given in a well-defined coordinate system. Thus, we do not need additional network components to deal with uncertainties in the coordinate frame. This leads to an overall reduction of complexity in the network, which allows us to work with a smaller amount of training examples. Additionally, a single labeled object of interest contains multiple local point neighborhoods, resulting in a multitude of available training examples. This greatly reduces the amount of labeled data which is needed to train the neural network.

The network is divided into three components. First, we extract a descriptive feature for the processed local point neighborhood. This feature then serves as input for a classification and a regression part of the network. The classification part classifies the local point neighborhood to be part of an object of a certain class of interest or not. In the latter case, it is assigned to an additional "not of interest" class. The regression part determines where the center of the object lies in relation to the processed local point neighborhood. This part of the network has to be trained separately for every object class of interest. Effectively, we end up with multiple regression sub-networks, one for every object class of interest.

3.3 Object recognition during post-processing

The output of the neural network is used to fill a voting space with weighted potential positions of objects of interest. To achieve this, we consider each local point neighborhood for which the classification part of the network results in a confidence level above a certain minimum, indicating a certain object class of interest. The regression output is then interpreted as 3D coordinate for the potential object position and transformed back into the original point cloud's coordinate frame. The weight of the positions is determined as follows:

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

where W_c = Weight of position for object of class c
$P(c)$ = Probability of or confidence for class c
n = Actual amount of points in neighborhood radius
s = Sub-sampling factor of point neighborhoods

n is used to normalize the weight according to the local point density. This should account for the problem that the density of MLS point clouds typically decreases with the distance to the sensor. It contains the actual number of points within the neighborhood's radius, i.e., the number of neighbors before the padding and random removal in the local point neighborhood's preparation step. The parameter s relates to the sub-sampling described at the end of Section 3.1. This parameter is not necessarily needed, but allows to change the ratio of processed local point neighborhoods without having to adapt other parameters like the detection thresholds.

Same as in our earlier approaches (Borgmann et al., 2018a), the voting space can be filled based on multiple sources of information. For example, it can accumulate the information within a multi-sensor system, i.e., the data of multiple sensors that operate in parallel. We use a rating process to decide if a potential position is regarded as a detected object's position. During this process, we find weight maxima in the voting space. This takes account of the fact that multiple local point neighborhoods vote for roughly the same position of an object if they actually belong to such an object, i.e., if this object exists at that position. In contrast, incorrect votes will not accumulate weight at a specific position.

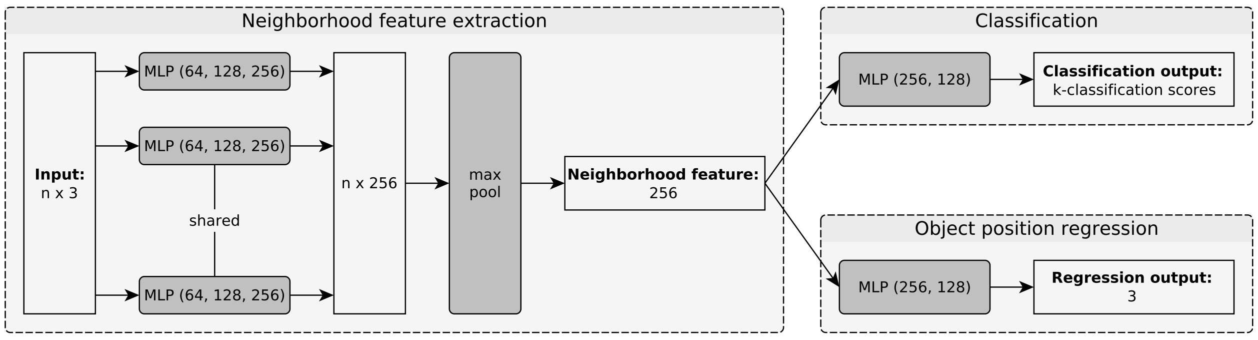


Figure 2. Chosen topology of the neural network. The network takes n 3D points as input and it outputs a classification score for k classes and a 3D coordinate of the expected object center. MLP stands for *muli-layer perceptron*. Batchnorm is used for all the MLP layers, except the output ones. We use dropout layers with a dropout rate of 0.2 in the classification and regression part of the network but not in the feature extraction part.

The rating process recalculates the weight of every potential position based on its current weight and the weight of other positions for the same object class in close proximity. It adds a certain percentage of the weight of the positions in proximity to the weight of the rated position. The percentage of weight added is defined by the normal distribution depending on the distance between the rated position and the position in proximity:

$$R_p = W_p + \sum_{k \in K} W_k \cdot e^{-\frac{{D_{pk}}^2}{2\sigma^2}} \tag{2}$$

where R_p = Rated weight of position p
W_p = Original weight of position p
K = Positions with same class as p in radius 2σ
W_k = Weight of position k
D_{pk} = Euclidean distance between positions p and k
σ determines the width of the normal distribution

After the recalculation of the weights, a threshold is applied and every position below that threshold is ignored. The remaining positions are regarded as detections of objects of the certain class. Since typically multiple positions for the same object remain, we find such clusters and merge the positions in each cluster in order to keep only one position per object.

4. EXPERIMENTS

We conducted several experiments to evaluate the capabilities of the presented approach and to quantify the influence of different parameter settings. In these experiments, we used LiDAR data recorded by a multi-sensor vehicle, and we focused on the detection of pedestrians. We have presented the measurement vehicle in detail in an earlier work (Borgmann et al., 2018b). In addition to multiple cameras, it is equipped with two Velodyne HDL-64E LiDAR sensors which provided us with the data for the experiments. The point clouds generated by both sensors are provided in the same common coordinate frame, which is achieved by GNSS/IMU-based direct georeferencing. In the context of the experiments, a single *point cloud* is defined as the data recorded by one LiDAR sensor within $0.1\,\mathrm{s}$, which quite accurately corresponds to one rotation of the sensor's scan head. The sensors perform 1.3 million measurements per second distributed over 64 scan lines. So each individual point cloud consist of 130,000 measurements. Since there are measurements in the direction of the sky, for instance, not all measurements provide meaningful results (3D points). The actual point clouds typically consist of around 110,000 points, depending on the vehicle's surroundings.

We used parts of two different data sets for the evaluation. One contains recordings of an urban environment with multiple pedestrians on sidewalks or street crossings. These data have been recorded around the site of the TUM (Technische Universität München) while the vehicle was driving in traffic[1]. The other one contains a staged sequence with a person moving between the fields of view of both LiDAR sensors of the vehicle. This second sequence was recorded at the site of Fraunhofer IOSB while the vehicle was either slowly moving or stationary. We performed a data fusion in the voting space while processing this second data set. For the evaluation of the results, pedestrians in both data sets have been labeled manually.

In order to train the neural network for all experiments, we used additional labeled point clouds which are disjoint from those used for the evaluation itself. In total, 1298 single point clouds were available for this purpose. Among these, 849 point clouds actually contain at least one pedestrian. The remaining ones were used as additional negative training examples. As explained in Section 3.2, every labeled object and every point cloud contains a multitude of local point neighborhoods, which increases the number of usable training examples. 20% of the available training data were set aside to validate the training progress. For the training we used an *Adam* optimizer (Kingma and Ba, 2014). The loss function for the classification part of the network was the sparse categorical cross entropy. For the regression part we used the mean squared error.

All three parts of the network have been trained in parallel. The classification part was trained using positive and negative examples. The regression part was trained using only positive examples. This means that we had more data available to train the classification part as compared to the regression part. In order to keep a balance in the amount of training done for both network parts, we randomly selected a subset of the available classification training data in each training epoch. The mean coordinate of all 3D points of a pedestrian was used as the object center

[1] http://s.fhg.de/mls2

while training the regression part of the network. The feature extraction part was trained alongside the classification and the regression part.

In our experiments, we included the optional removal of ground points. Besides this, no additional segmentation steps were performed. We also used the optional sub-sampling mentioned at the end of Section 3.1 with $s = 5$, meaning we only generated and processed a local point neighborhood for every 5th point of a point cloud.

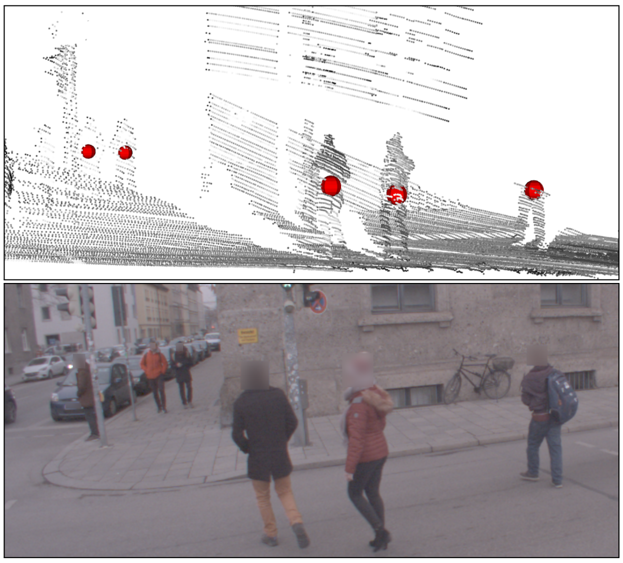


Figure 3. **Top:** An example of the output of our approach. **Bottom:** For comparison an image which shows the same situation. Five of the six visible pedestrians are detected. The missing one is partially occluded by a traffic light post.

The output of our approach are the positions of recognized pedestrians. An example of such an output is shown in Figure 3. Results like this are compared to ground truth data to determine true positive, false positive and false negative recognitions. These values are then used to determine *precision* and *recall*, which are defined as follows:

$$Precision = \frac{tp}{tp + fp} \tag{3}$$

$$Recall = \frac{tp}{tp + fn} \tag{4}$$

where tp = True positive
fp = False positive
fn = False negative

For every configuration we performed multiple runs with different recognition thresholds. The results of these multiple runs can be plotted as a precision-recall curve.

4.1 Neighborhood parameterization

The first set of experiments covers different parametrizations to define the local point neighborhoods. This includes the three main parameters:

1. The radius of the sphere around the central point which is considered as the local point neighborhood.
2. The required minimum number of points in the neighborhood radius.
3. The target number of points in the neighborhood, which is reached by random selection or padding.

We conducted experiments for each of these three parameters. The results shown in Figure 4 are discussed in the following paragraphs.

Neighborhood radius The radius is probably the most significant parameter to define the local point neighborhoods. It greatly influences the presented approach in different ways. In an ideal case, each local point neighborhood only contains points which are part of one specific object. If the neighborhood radius is too large, this is rarely the case if multiple objects are in close proximity to each other. This effect can decrease the detection and recognition performance. On the other hand, a neighborhood with only a small radius does not contain enough information to allow for good results.

We compared the results of choosing a neighborhood radius of $0.2\,\mathrm{m}$, $0.3\,\mathrm{m}$ and $0.4\,\mathrm{m}$. For these experiments the parameter for the minimum number of points in the neighborhood radius was set to 10 and the one for the target number of points to 100. As can be seen in Figure 4a, a neighborhood radius of $0.2\,\mathrm{m}$ is too small to achieve usable results. With a radius setting of $0.3\,\mathrm{m}$ or $0.4\,\mathrm{m}$, the results are acceptable and quite similar.

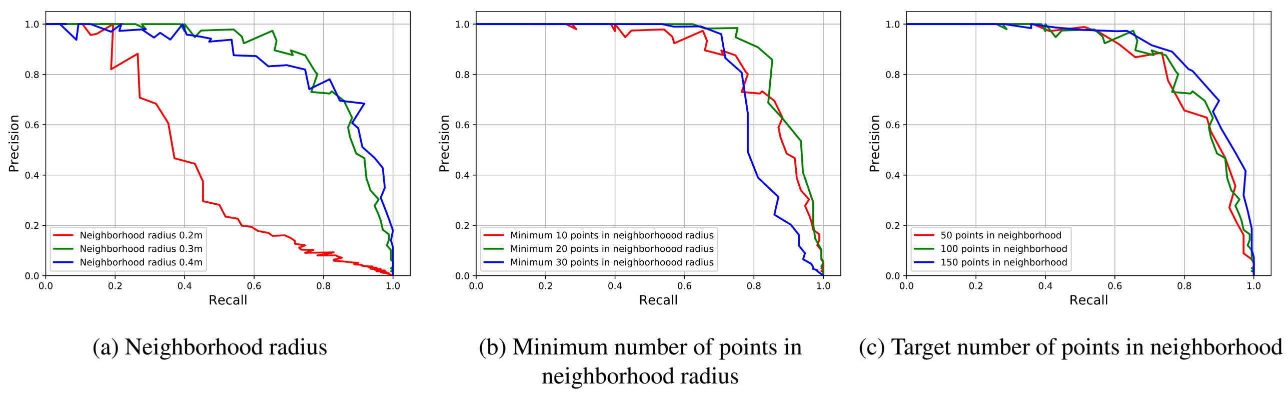
Minimum number of points in neighborhood This threshold acts as a lower bound for the local point density. If the density is too low, there are generally not enough points in the neighborhood to reach this threshold. This primarily affects the performance in high distances to the LiDAR sensor, where the point density is low and prevents any recognition of objects. This has a negative impact on the recall rate. Since local point neighborhoods that contain only a few points do not include much information, the precision decreases if the threshold for the minimum number of points is set too low.

The results shown in Figure 4b are consistent with these expectations. We compared values of 10, 20 and 30 for this parameter and achieved best results with a setting of 20. For these experiments we set the radius to $0.3\,\mathrm{m}$ and chose 100 as the target number of points in a local point neighborhood.

Target number of points in neighborhood The target number of points is a parameter which is necessary since the neural network requires a fixed input size. Generally speaking, if this parameter is too low, relevant information gets lost. If its too high, the target number of points is in most cases achieved by padding. These additional points provide no additional information, but still have an impact on the run-time, which is not desirable. The results shown in Figure 4c where achieved using a neighborhood radius of $0.3\,\mathrm{m}$ and a setting of 10 as the minimum number of points in that radius.

4.2 Number of training examples

One advantage of our approach is its low demand for training data. We tested the limits of this capability by reducing the number of examples during training. For this experiment we



(a) Neighborhood radius (b) Minimum number of points in neighborhood radius (c) Target number of points in neighborhood

Figure 4. Precision-recall curves for different parametrizations of the local point neighborhoods

used a neighborhood radius of $0.3\,\mathrm{m}$, 20 as the minimum number of points in that radius and 150 as the target size of the neighborhood. We trained three networks with this configuration: One used all 1298 point clouds available for its training, the other one only 500 randomly selected training point clouds, and the third one only 100 which were randomly selected out of the previous 500. For all three networks, we still used 20% of their respective training data for validation purposes.

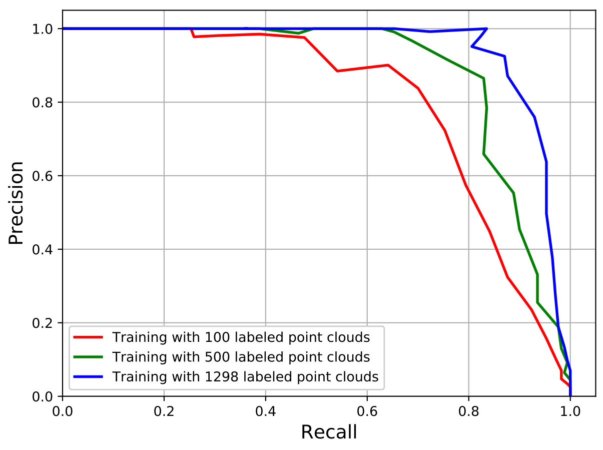


Figure 5. Results for different numbers of labeled point clouds used for training

The results of this experiment are shown in Figure 5. As expected, a smaller amount of training data has a negative impact on the performance of the proposed method. At least for the case of only using 500 labeled point clouds, we are still able to achieve acceptable results. In case of only using 100 labeled point clouds for training of the network, the results are still usable, but it may be beneficial to reduce the complexity of the neural network. Generally speaking, the network complexity should scale with the amount of available training data.

4.3 Sub-sampling factor

At the end of Section 3.1 we described a simple sub-sampling which allows us to reduce the amount of local point neighborhoods to be considered. This reduction is meant to reduce the processing time. For the previous experiments, this parameter was set to 5, meaning for every 5th point a local point neighborhood has been generated and analyzed.

In this experiment, we want to quantify the effect that the sub-sampling has on the quality of the results. We tested four different settings for this parameter: 3, 5 (our previous baseline), 10 and 20. For the other settings, we used a neighborhood radius of $0.3\,\mathrm{m}$, 20 as the minimum number of points in that radius and 150 as the target size of the neighborhood.

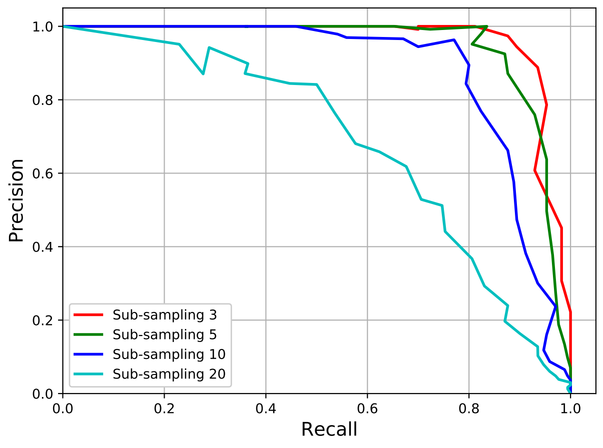


Figure 6. Results for different settings of the sub-sampling parameter.

Figure 6 shows the results of this experiment. While lower values for the sub-sampling only have a small impact on the results, the impact increases significantly if this parameter is set to 10 or 20. This can be explained by the point density and the amount of points for a pedestrian in the data. Especially in larger distances, the point density is low and a pedestrian is typically represented by less than 100 points. In case of a sub-sampling parameter of 20, approximately only 5 local point neighborhoods are generated for such a pedestrian. Under these conditions, the potential influence of even a single neighborhood for which the neural network outputs a wrong result is high and leads to a false positive or a false negative detection.

Generally speaking, the maximum sub-sampling has to match the data density in the processed data. If this density is high, higher values for the sub-sampling are possible without reducing the quality of the results. In addition, the potential benefit with regard to the processing time is higher in such cases.

5. CONCLUSION AND FUTURE WORK

We presented a novel approach which uses a neural network to process the 3D coordinates of points in local point neighborhoods, and which combines the output of this neural network with an ISM-like voting process to detect, recognize and locate objects of interest. The usage of local point neighborhoods is inspired by classical feature extraction methods for point clouds which often generate per-point features based on their relation to neighboring points.

Processing local point neighborhoods in a neural network has several benefits: On the one hand, the local point neighborhoods are comparatively small allowing for less complex neural networks which also require less training data. On the other hand, they are able to generate local features and classifications without having to rely on a previous explicit segmentation. The ISM-inspired voting process allows to overcome the low information content of the local point neighborhoods by combining the results of multiple of such neighborhoods.

We evaluated the presented approach in a set of experiments which were focused on the detection of pedestrians and tested the influence of certain parameters on the results. We also quantified the limits of the approach with regard to the amount of required training data and were able to achieve good results even with very limited amounts of training data.

In future work we want to add a hierarchical object recognition to our approach which not only recognizes objects themselves, but also locates certain parts of the objects. Such parts of objects can constitute a suitable source of information for an enhanced situation analysis. In the example of pedestrians, relevant body parts could be hands, feet and head. In addition, we want to add a tracking component to our approach to be able to keep track of detected objects over multiple point clouds in a sequence.

References

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

Borgmann, B., Hebel, M., Arens, M., Stilla, U., 2017. Detection of persons in MLS point clouds using implicit shape models. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* XLII-2/W7, 203–210. https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2-W7/203/2017/.

Borgmann, B., Hebel, M., Arens, M., Stilla, U., 2018a. Usage of multiple LiDAR sensors on a mobile system for the detection of persons with Implicit Shape Models. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*,XLII-2, 125–131. https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2/125/2018/.

Borgmann, B., Schatz, V., Kieritz, H., Scherer-Klöckling, C., Hebel, M., Arens, M., 2018b. Data processing and recording using a versatile multi-sensor vehicle. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-1, 21–28. https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-1/21/2018/.

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. In: *2016 International Joint Conference on Neural Networks (IJCNN)*, 1578–1584.

Johnson, A.E., Hebert, M., 1999. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5), 433–449.

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

Maturana, D., Scherer, S., 2015. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In: *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. http://ijr.sagepub.com/content/29/12/1516.abstract.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017a. PointNet:Deep Learning on Point Sets for 3D Classification and Segmentation. In: *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. In: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (eds), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 5099–5108.

Rusu, R.B., Blodow, N., Beetz, M., 2009. Fast Point Feature Histograms (FPFH) for 3D Registration. In: *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, IEEE Press, Piscataway, NJ,USA, 1848–1853.

Rusu, R.B., Marton, Z.C., Blodow, N., Beetz, M., 2008. Learning Informative Point Classes for the Acquisition of Object Model Maps. In: *2008 10th International Conferenceon Control, Automation, Robotics and Vision*, 643–650.

Socher, R., Huval, B., Bath, B., Manning, C.D., Ng, A.Y., 2012. Convolutional-Recursive Deep Learning for 3D Object Classification. In: *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. http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/I-3/179/2012/.

Weinmann, M., Jutzi, B., Hinz, S., Mallet, C., 2015. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105, 286-304.