F O U N D A T I O N S O F C O M P U T I N G A N D D E C I S I O N S C I E N C E S
Vol. 50 (2025) No. 3
ISSN 0867-6356
sciendo DOI: 10.2478/fcds-2025-0013 e-ISSN 2300-3405

Annotation-free Generation of Training Data Using Mixed Domains for Segmentation of 3D LiDAR Point Clouds

Konrad Cop[1,2], Bartosz Sułek[2], Tomasz Trzciński[1]

Abstract. Semantic segmentation is important for robots navigating with 3D LiDARs, but the generation of training datasets requires tedious manual effort. In this paper, we introduce a set of strategies to efficiently generate large datasets by combining real and synthetic data samples. More specifically, the method populates recorded empty scenes with navigation-relevant obstacles generated synthetically, thus combining two domains: real life and synthetic. Our approach requires no manual annotation, no detailed knowledge about actual data feature distribution, and no real-life data of objects of interest. We validate the proposed method in the underground parking scenario and compare it with available open-source datasets. The experiments show superiority to the off-the-shelf datasets containing similar data characteristics but also highlight the difficulty of achieving the level of manually annotated datasets. We also show that combining generated and annotated data improves the performance visibly, especially for cases with rare occurrences of objects of interest. Our solution is suitable for direct application in robotic systems.

Keywords: deep learning for visual perception, semantic segmentation, 3D LiDAR, robotic perception, point clouds

1. Introduction

In robotic 3D perception, extracting semantic information about objects present in the environment is crucial to make relevant navigation decisions. This task can be achieved by solving the semantic segmentation problem of 3D perception data (point

*[1]Warsaw University of Technology, Faculty of Electronics and Information Technology Nowowiejska 15/19, 00-665 Warsaw, Poland
[2]United Robots Sp. z o.o., Świeradowska 47, 02-622 Warsaw, Poland
Corresponding author: cop.konrad@gmail.com

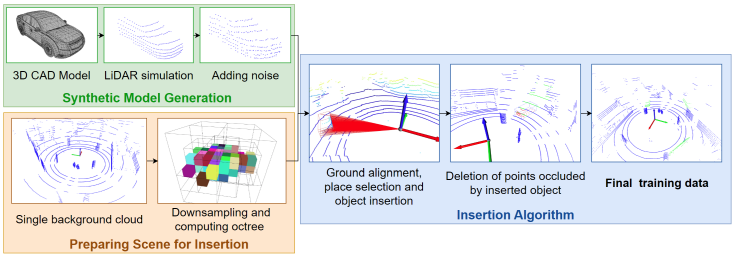

Figure 1. Process of training data generation through the integration of synthetic 3D objects into recorded background point clouds

clouds). In such a case, the segmentation problem is usually defined as classifying each point into one of multiple distinct categories [28]. In this paper, we reformulate the problem in a robotic context by focusing solely on classifying specific obstacles that block the robot's path while treating the remaining points as nonessential. In mobile robot operation, the efficiency deteriorates the most when the robot must diverge from the intended path due to unexpected obstacles [17, 3]. In such a case, classifying obstacles can help the robot make correct navigation decisions while distinguishing the objects that always remain static (fixed), e.g., walls, pillars, furniture, etc., is not equally relevant. In other words, our task is to perform a binary semantic segmentation of a point cloud to distinguish objects of interest from the background. The proposed formulation emerges from the authors' experience with the deployment of industrial floor scrubbers in various environments. A UR\Cleaner [22] robot requires point cloud segmentation capability to adjust its path when working in a floor coverage mode.

This problem can be solved by Deep-Learning-based segmentation networks but requires vast training datasets. In the classical approach, these datasets are generated by manual annotation of data samples. This is especially time-consuming for 3D LiDAR data and becomes prohibitively tedious due to the variety of possible environments [15], as in practice, each environment requires separate annotation. A directly correlated issue might be the infrequency or, in extreme cases, lack of instances of the object of interest in the dataset to train the model sufficiently. Therefore, when deploying a robot in a new environment one has a choice either to use available, annotated datasets, which might have a different feature distribution, or to perform a new annotation.

Another way to generate vast training datasets utilizes simulation frameworks in which any "virtual world" can be created from digital models of objects and infrastructure. In such a "world" it is possible to model sensor characteristics (e.g. resolution and FoV) to perceive surroundings and generate virtual LiDAR scans [9, 26]. Although it is free from manual annotation, it suffers from the need to laboriously (manually) generate the complete appearance of the environment using 3D CAD models. What is more, such synthetic models are not necessarily available, especially for unstructured objects such as vegetation or various building installations. Additionally, the synthetic representation experiences a reality gap originating from the

inability to fully model complex noise characteristics of sensors' perception of real objects and their small details.

To address the limitations of these two approaches, we propose a method that combines the data from two mixed domains, namely real-life (recorded by a sensor) and synthetically simulated. More specifically, we use real 3D LiDAR scans obtained in the environment, which we refer to as *background scans*, and augment them with synthetically generated instances of the objects of interest, utilizing a multi-step process. The scans of objects are generated using the simulation framework Gazebo [16], through the observation of single CAD models with a simulated LiDAR at various distances and orientations. Next, each instance point cloud undergoes a noise addition process consisting of various techniques. Finally, the instance is placed within the real-life (recorded) background scan in a random but likely position. The object gets directly annotated upon insertion in the scene, while the background scans are classified as "unknown". As a result, an automated procedure of point clouds generation for training is achieved, which requires no additional manual work. Our system assumes that the background scans are available, which is usually the case when a robot is deployed in a new environment and the navigation map is created. Additionally, it requires no detailed knowledge about the characteristics of the actual dataset but utilises basic observations of the robot data and leverages purely statistical methods. The system allows the creation of meaningful training datasets from scratch for unlimited types of objects, provided that proper synthetic (CAD) models are available.

To verify our approach we collected data in an underground parking environment and augmented them with synthetically generated car instances. Subsequently, we trained the semantic segmentation network with these data to validate the performance. What is more, when applying segmentation in real, industrial scenarios, one would first use available datasets from similar use cases to validate their applicability. We conducted such experiments with well-known, open-source datasets rich in car instances, i.e. Semantic KITTI [4] and Pandaset [27], which we used in original and converted form. To validate all experiments, we created a ground truth dataset coined URset by manual annotation of collected data. The results indicate that datasets generated by our system perform better than the open-source datasets but are not able to match the performance of a manually annotated dataset. We also show that merging annotated and synthetically generated sets helps in achieving better Accuracy and IoU, especially in situations where it is difficult to obtain a significant amount of data due to the rare occurrence of objects of interest.

The following sections describe in detail the process of dataset generation, the comparison with open-source datasets, and the discussion of experiments.

This article is a substantially extended version of a short conference paper presented at the 5th Polish Conference on Artificial Intelligence PP-RAI 2024 [7]. Compared to the original publication, it contains an additional literature review, an extended description of data generation and noise injection strategies, and details about various datasets. It also differs in terms of experiments, specifically, the detailed analysis of the influence of noise injection strategies and comprehensive Ablation Study.

2. Related work

The generation of training data can be split into three main approaches: fully manual or semi-automated annotation, simulation of sensor returns in virtual worlds (fully synthetic data), and real data augmentation with synthetic data.

The first approach is a classical way to obtain training data in Machine Learning and requires a human in the loop performing annotation. There are multiple tools available specifically for point clouds, both open-source, e.g., Point Labeler [4], and proprietary, like CVAT [8] or Segments.ai [21], where a user has various features to select from. Usually approaches like bounding boxes or individual points painting are used. To accelerate the process, numerous semi-automated methods were proposed. Meng et al. [20] introduced a weakly supervised approach where a user points to inexact positions of objects. Arief et al. [2] proposed to utilize tracking of clusters' motion to deduct annotation in consecutive scans. Qian et al. used 2D boxes to generate precise 3D box annotations utilising Transformer architecture. Multiple similar approaches were proposed and despite significant time reduction, the common characteristic of these methods is the requirement of at least a limited annotated set.

The second approach is fully based on synthetic dataset generation by simulating sensors in a digital environment. Multiple teams tackled this problem, and the usual differentiation is the simulation framework used and the scale of the virtual world. Hurl et al. [14] utilised a world obtained from the GTA V video game to generate 50.000 frames. Xiao et al. proposed SynLiDAR, a large-scale dataset consisting of 198.396 point clouds obtained by means of Unreal Engine 4 from multiple virtual worlds. The tests on both datasets show promising results however, the creation of realistic and detail-rich models was extremely time-consuming and required professional knowledge in architectural modelling and experience in virtual reality.

Similarly, for the third group of algorithms, numerous works were proposed. In [25] authors proposed point cloud augmentations fusing 2D camera images with 3D LiDAR points. The authors of [6] propose a set of strategies to locally augment point cloud regions. Both algorithms improve the performance of segmentation but operate on real-life, previously annotated datasets. Other researchers proposed to use knowledge learned from limited real-life data to inject noise into simulated point clouds [19], [10]. The most similar concept to ours was proposed by Fang et al. [11], with the difference that the authors select instances' insertion places using learned distributions, which ultimately requires a labeled dataset. Our approach relies solely on statistical and geometrical methods for insertion, and no additional dataset is necessary.

3. Data generation

This section describes in detail the process of creating mixed data samples consisting of synthetic instances of objects of interest and real-life background scans. The procedure consists of three main parts described in the following section, namely generation of synthetic instances, insertion into a background point cloud, and augmenting with

Figure 2. Example of a ShapeNet car 3D CAD models used for data generation.

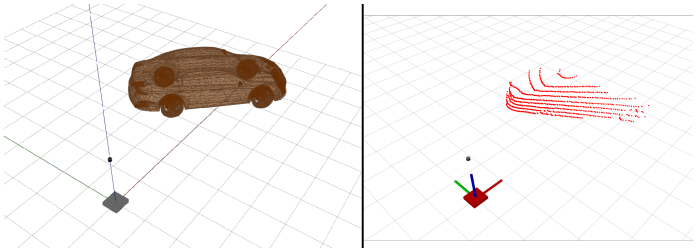

Figure 3. (Left) Original 3D CAD model of a car in Gazebo; (Right) Simulated LiDAR point cloud

simulated noise. The whole generation process is shown in Figure 1, which explains how the algorithm generates a mixed-domain dataset by inserting multiple cars per scan.

3.1. Synthetic object generation

Our use case focuses solely on *cars* segmentation therefore, a collection of appropriate 3D CAD models was required for use in a LiDAR simulation. After evaluating various datasets, ShapeNet [5] was selected due to its large-scale repository. ShapeNet contains 3,515 models categorized in the general *"car", "auto", "automobile"* labels. For this specific task of segmenting cars in public spaces, we focused on the *Car, Sedan, Coupe,* and *Hatchback* subclasses, excluding ambiguous models such as sports cars and limousines, which are not representative of vehicles typically found in deployment environments. The remaining models were manually reviewed to remove corrupted instances, resulting in a curated set of 415 valid car models. Some examples of car instances are shown in Figure 2.

Before using the filtered dataset, additional scaling adjustments were applied to ensure realistic car dimensions. ShapeNet models often exhibit unusual proportions. The provided scale factors did not always produce real-world sizes. To address this, the car models were rescaled by sampling their heights from a uniform distribution between 1.55 and 1.7 m, based on real-world observations.

The scaled models were used in LiDAR simulations to generate single-point cloud scans. We utilized the Gazebo simulation framework, where each car model was

inserted sequentially. For each car instance, multiple scans were generated by applying Z-axis rotations and using different distances from the LiDAR sensor. The distances were sampled from a Gaussian distribution of mean μ_d and standard deviation σ_d. The varying range and the LiDAR data sparsity should reflect real-world conditions to avoid overfitting during model training to specific distances.

For scan generation the Gazebo plugin for the Velodyne VLP-16 LiDAR sensor was used, replicating the exact hardware specifications of the robot. The simulation requires specifically a proper number of channels and horizontal resolution as described in Table 1. This process is depicted in Figure 3.

3.2. Object insertion into background scans

During the deployment and operation of the robot, a significant number of LiDAR scans can be recorded and utilised for dataset creation. Instead of simulating complete, detailed environments for each deployment location, we leverage the collected data, which contains detailed information about each scene, and we use it for background creation. The ultimate combination of real and synthetic data ensures that the generated datasets remain realistic. It is important to notice that in our system, each scan is treated separately, so the translational or angular displacement of the object between consecutive scans is not relevant. An object that moves during the scanning might appear stretched in the recorded point cloud. However, the scans are obtained from the sensor at a working frequency of 20 Hz, which results in around 50ms needed for a complete 360° scan. Considering that an object usually occupies a fraction of the angular Field of View, we assume that we can treat the object in each scan as static, therefore, we did not introduce any motion compensation.

3.2.1. Background scans preprocessing

The background scans are simply individual point clouds, i.e., a single return of a LiDAR sensor recorded in the environment without objects of interest. Consequently, they were preprocessed to serve as a robust base for synthetic object insertion, as shown in Figure 4. The process consists of the following steps:

- **Octree structure creation**: An octree representation of the background cloud to accelerate spatial queries, such as collision checks, during object insertion.
- **Downsampling**: Voxel downsampling applied to the cloud and its corresponding octree representation for further optimization of collision detection and occlusion checking.
- **Ground plane detection**: The ground plane is identified with the RANSAC algorithm, fitting a plane to the lowest points in the downsampled scan. This ensures that inserted objects align correctly with the ground, preventing issues like floating or floor penetration.

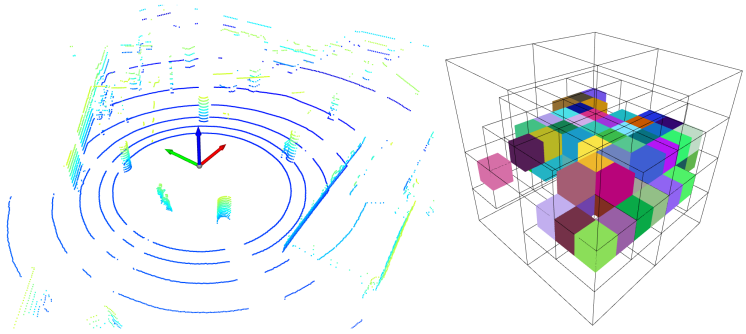

Figure 4. (Left) Example background cloud; (Right) Octree structure of background cloud

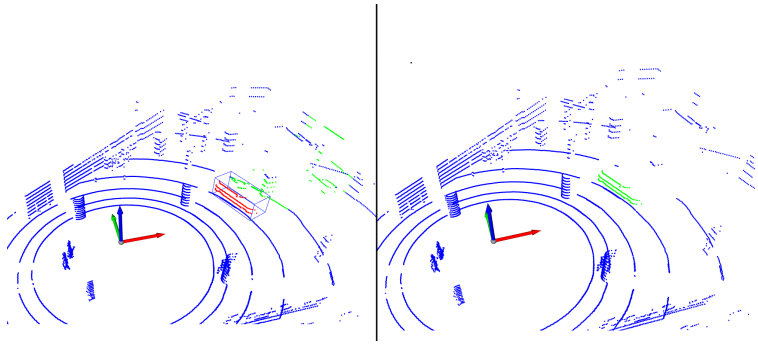

Figure 5. (Left) Detection of points occluded by inserted object; (Right) Scene after removal of occluded objects

3.2.2. Object insertion algorithm

The object insertion algorithm must ensure realistic object placement, addressing the reality gap introduced by synthetic data. It is important to note that the insertion process relies only on basic real-world observations, without utilizing any information about already gathered and labeled data. Specifically, it does not leverage additional statistics about specific objects' placement or distributions. The number of car instances n_i to be placed in each background scan is sampled from a Gaussian distribution with a mean μ_n and standard deviation σ_n. We then pick n_i random synthetic instances generated in the previous step and try to insert them into the scene (provided that the insertion criteria described below are fulfilled). This approach introduces various random numbers of locations in different scenes. Synthetic cars are inserted sequentially (one after another), with each instance at an already fixed distance from the LiDAR, which was defined upon the generation of a synthetic object as described in Section 3.1. The insertion distance is, therefore, the same as the Gazebo generation distance. Only Z-axis rotations are varied to find suitable

insertion candidates. The steps of the object insertion process are as follows:

1. **Transformation computation**: the synthetic object is transformed into the coordinate frame of the background cloud LiDAR. Initial random rotation along the Z-axis is applied to introduce orientation variability.

2. **Ground alignment**: the object is aligned with the detected ground plane by its bounding box corners adjustment, ensuring proper placement without floating or floor penetration.

3. **Collision and occlusion checks**: the algorithm verifies the following conditions:

   - if the object does not collide with the background environment. Other points within the car's bounding box are counted, and collision is detected if a specified threshold of points number is exceeded.
   - If the object does not collide with any previously inserted synthetic objects.
   - If the object is not heavily occluded by the other environment elements, that would prevent LiDAR from capturing it. This is checked by tracing lines from the LiDAR to potential obstacles.
   - If the object does not occlude or become occluded by other inserted synthetic objects. Excessive occlusion is avoided by limiting overlap between objects.
   - If the object is not inserted into empty spaces, e.g., outside the background scan bounds.

   Unless any aforementioned conditions are met, the algorithm attempts the insertion of the same object at a new Z-axis rotation angle. The new rotation angle is calculated by adding 10°to the previous angle, ensuring that the entire 360°scope around the LiDAR is checked for suitable placement in increments of 10°. If the object cannot be inserted at any of the 36 possible positions, a different synthetic object is selected for the placement search. Additionally, if k_f consecutive objects fail to be inserted into the scene, the algorithm terminates the insertion process, to ensure that it does not stuck in an infinite loop of failed placements.

4. **Occlusion detection**: Background points occluded by the inserted synthetic object are removed. Using the octree structure, points within the cone defined by the LiDAR position and the object's bounding box are marked. Rays are traced from the LiDAR to these points, and any intersection with the bounding box results in occlusion removal, as shown in Figure 5.

The four-point approach described above ensures that the inserted objects blend seamlessly with the background scans, creating diverse training datasets suitable for semantic segmentation tasks. It's worth mentioning that the insertion algorithm will also work with real-world instances, e.g., car scans extracted from different open-source datasets. Note that, after checking various insertion conditions, the original

distribution of distances of cars will be different as some samples will be skipped from insertion.

3.3. Noise injection strategies

After generating complete scenes with inserted synthetic cars, the next essential step is to reproduce the noise characteristics of real sensor data. It is important to notice that the method utilizes no knowledge about the type of surface of objects of interest, and therefore, no noise characteristics can be assumed for various elements of the object (e.g., different shininess of windows vs wheels). Instead, we advocate for purely statistical methods to add noise. We propose the following noise injection techniques to apply to synthetic cars to simulate imperfections typically observed in LiDAR data. These methods enhance the generalization of the training datasets by introducing diverse perturbations.

- **Dropout**: single points are randomly removed from object clusters to simulate data sparsity and reduced resolution, reproducing data loss during transmission. The probability of dropout for each point was sampled dynamically from a uniform distribution up to 0.3% chance.
- **Gaussian noise on each cluster**: we observe that the biggest fluctuations in real data happen along the direction of LiDAR channels therefore, we propose a LiDAR-oriented Gaussian noise applied to the points in each object instance by adding a small factor to the range sampled from a zero-mean Gaussian distribution. During data generation in Gazebo, a standard deviation sampled from uniform [0.001, 0.03] was used.
- **Random points injection**: some specific objects like cars often lack interior details visible through windows, as most 3D CAD models do not include interior parts. To address this problem, we propose to inject random points into each car cluster to simulate interior noise. We propose to generate a random number of points from the range [0,15], each point placed on a virtual sphere according to polar coordinates at radius generated using a normal distribution ($\mu = 0.3$, $\sigma = 0.5$), azimuth and elevation were uniformly sampled angles.
- **Random points class modification**: a small random subset of 2% of generated scene points was assigned incorrect class labels. This step simulates labeling errors that may occur during manual annotation.
- **Cut-off of random clusters**: To simulate the occlusion of various objects, we also apply a vertical cut-off, which effectively removes a part of an object before insertion in the scene. There is a 60% chance during synthetic object selection that a chosen car instance would be vertically truncated. The truncation was limited to between 20% and 80% of the input points. This step simulates very frequent occlusions caused by other objects or blind spots, representing scenarios where objects are partially visible.

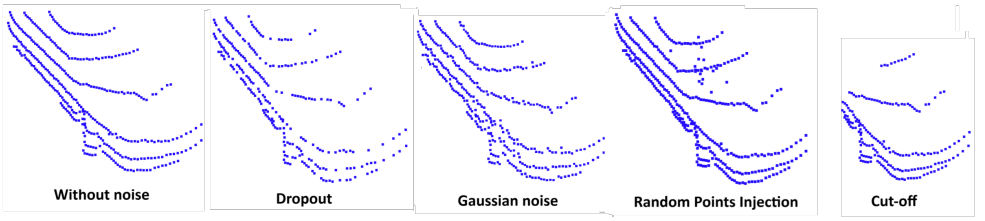

Figure 6. Used noise injection strategies.

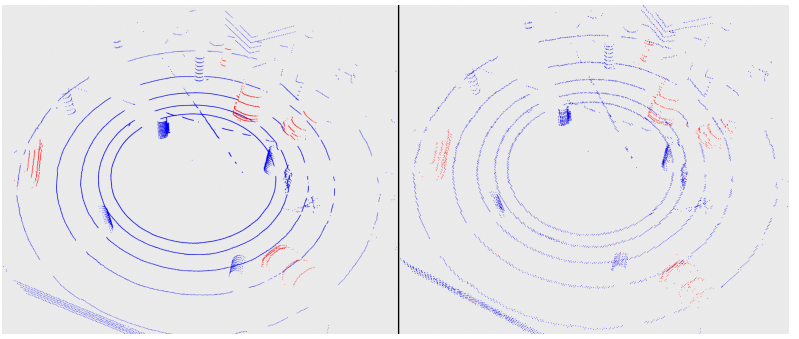

Figure 7. (Left) Generated scene before adding noise and augmentations; (Right) Generated scene after adding noise and augmentations.

The combined use of these noise injection strategies ensures that the segmentation model is trained on data that closely mirrors real-world environments, which significantly influences the model training process and results. Subsequent steps of noise injection on an exemplary cloud are shown in Figure 6, and the results of augmentation are depicted in Figure 7.

3.4. General data augmentation strategy

Training a deep learning model on partially synthetic data with the goal of deploying it and inference on real-world data requires the model to possess high generalization capabilities. This is essential to avoid overfitting to synthetic data, which, despite using various noise modeling techniques, are highly different from real-world data. To address this, in addition to the noise injection, various dataloader-level augmentation methods were applied, ensuring the dataset reflects a wide range of real-world conditions and processing errors. These augmentations simulate diverse variations in training data, which may originally reflect many similarities due to the use of scans from a limited number of robot recordings. Below are the augmentation strategies implemented in the data loader:

- **Random rotation**: A random rotation around the Z-axis is applied to the entire scene. The rotation angle is sampled uniformly over 360 degrees.

- **Random flipping**: The point cloud is randomly flipped along the X-axis, Y-axis, or both axes, introducing symmetry variations.
- **Scaling**: Random scaling factors, sampled uniformly from a range [0.9, 1.1], are applied to the X and Y coordinates of the points.
- **XYZ Translations**: XYZ-shift augmentations were applied to both car object instances and the entire map, with shifts sampled from uniform distributions [1]. For each full scan sample, a random translation vector is sampled from: [-5, 5] for the X and Y coordinates, and [-2, 2] for the Z coordinate.
- **Elastic Deformation**: Elastic deformations were used to introduce localized distortions to point clouds, simulating different imperfections caused by environmental factors. A random 3D noise field, smoothed with several 3D convolutions and interpolated for continuous displacement, deforms a point cloud with a granularity parameter controlling resolution and a magnitude parameter determining deformation intensity. This strategy was proposed by authors of SphereFormer [18].

In addition to the aforementioned geometric transformations, standard feature-space normalization techniques are applied. Point clouds are voxelized into a uniform grid of a size [0.1, 0.1, 0.1] to normalize their density, with unique points selected within each voxel. This ensures consistent spatial representation and reduces the training dependence on input data sparsity.

4. Datasets

This chapter describes various datasets that we have used throughout our experiments. As mentioned before, the classical way to generate training datasets is to perform manual annotation on real-life data. In this way, we created the first dataset for experiments, which is a rich set of LiDAR point clouds, where all car instances were manually annotated, coined **URset**. This dataset is described in detail in the section 4.1.

When however, one aims to avoid manual labour, a typical way is to reach for already available datasets of identical or at least similar characteristics. We followed that path too, and in the absence of underground parking datasets, we focused on sets that contain numerous car instances and are obtained by a rotating LiDAR [12]. To this end, we decided on the well-known Pandaset [27] and Semantic KITTI [4], which, however, require additional conversion to be usable in the proposed scenario. The details are described in Section 4.2.

Finally, the last section (4.3) of this chapter focuses on a description of the dataset, which was created using the methods proposed in this paper. We refer to it as **URset_syn**. A visual comparison of various datasets is depicted in Figure 8, and the numerical data are gathered in Table 1.

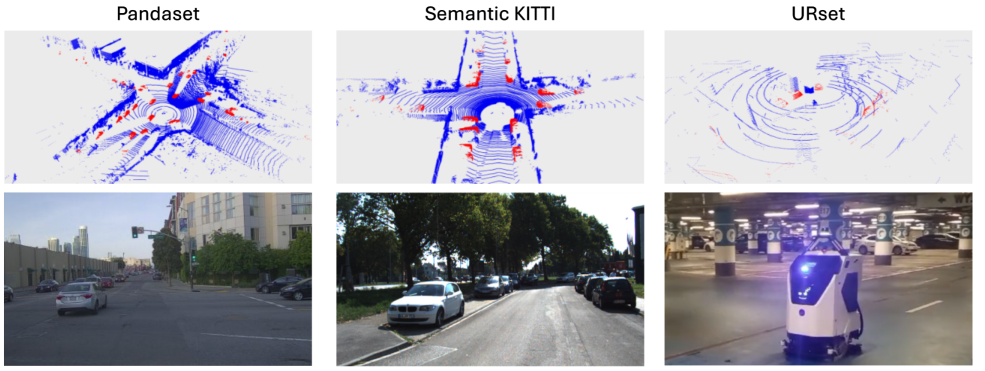

Figure 8. Comparison of appearance of analysed datasets and exemplary point clouds. There are clearly visible differences between URset and open-source sets, such as the sparsity of point clouds (16 vs 64 LiDAR beams), the frequency of occurrences of cars, and the different infrastructure of the environment.

	Feature	Pandaset	Semantic KITTI	URset	URset_syn
Data Specs	# annotated point clouds	6,080	23,201	6,670	6,740
	# scenes	103	22	8	3
	Annotation time [hrs]	unknown	1700*	32*	N/A
LiDAR Specs	Sensor model	Hesai Pandar64 [13]	Velodyne HDL-64E [23]	Velodyne VLP-16 [24]	Velodyne VLP-16 [24]
	Range [m]	200	120	100	100
	Vertical FoV [°]	-25 to +15	-24.9 to +2	-15 to +15	-15 to +15
	# channels	64	64	16	16
	Mounting height [m]	2.1	1.73	1.51	1.51
	Horizontal resolution [°]	0.2	0.09	0.4	0.4

Table 1. Comparison of parameters across datasets. *For Semantic KITTI, annotation resulted in complete point cloud labels; for URset, only objects of interest (cars) were labeled.

4.1. URset

To generate this set, we utilised localisation data obtained by a cleaning robot in an underground parking environment. The alignment of the consecutive scans was obtained using a UR-based SLAM approach. Overall, it consists of 6,670 scans containing 46 593 car instances, divided into train, validation, and test subsets in the proportion 70/15/15%. The dataset was split into subsets based on unique sequences recorded in different locations of the environment to prevent data leakage. One should understand a "sequence" as a series of scans recorded in a specific area of a map. The details are depicted in Figure 9. The training set consists of 6 sequences, while the

validation and test sets each contain 1 sequence with a varying number of scans. The manual annotation of all sequences took 32 hours. In all following experiments, the test subset of URset was used to validate various dataset configurations.

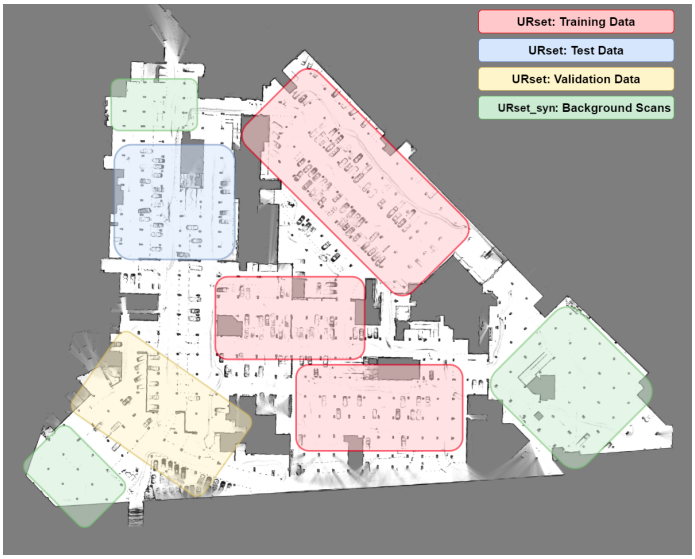

Figure 9. Example of area selection for dataset creation. Each area corresponds to a recorded sequence, with different sequences representing specific regions of the environment. Background scans used for synthetic data generation were recorded in areas without any cars present.

4.2. Open Source Datasets

Faced with an unannotated set of point clouds, it is difficult to judge the similarity of various datasets to make a proper choice and the only way is to perform qualitative analysis. The comparison of various features of these sets is listed in Table 1. As clearly visible there, the characteristics of a dataset are tightly coupled with the hardware used for recording. Specifically, aspects such as the number of channels (beams), vertical and horizontal resolution, Field of View, or altitude of the sensor are unlikely to be identical. The visual comparison between the datasets is shown in Figure 8. Both Pandaset and Semantic KITTI are using high-quality LiDARs, Hesai Pandar64 and Velodyne HDL-64E respectively. These sensors have much higher density and resolution of data than our low-cost, 16-channel Velodyne VLP-16. Additionally, the Field of view differs significantly, which results in different parts of the environment being perceived. What is more, looking at the photos of the environment, a clear difference between the background infrastructure can be noticed as well as the orientation of objects (how the cars are parked). All these aspects not only

need experimental verification but also raise concerns about the applicability of the data, therefore, we propose a strategy to unify the datasets.

To address the discrepancies originating from hardware differences, we propose a set of conversion steps on the point clouds recorded by the Pandar64 and HDL-64E. It is important to notice that one cannot simply pick every 4^{th} beam from dense 64-channel LiDAR and treat it as a beam of a 16-channel LiDAR. Instead, we propose to transform both datasets in a way that simulates proper horizontal resolution (0.4 m), number of channels (16), vertical FoV $(-15°, 15°)$, and mounting of the LiDAR (1.51m). The following steps were taken to achieve this:

- Projecting points from Pandaset and SemanticKITTI onto 16 conical surfaces based on their distance from the nearest surface. The sensor is placed in the origin of the point cloud $z = 0$.
- Scanning each of the 16 simulated surfaces angularly at 0.4° intervals, matching the horizontal resolution of the VLP-16. From each scanned region the closest point is selected, removing all points located behind it, simulating the channel intersections with object surfaces.

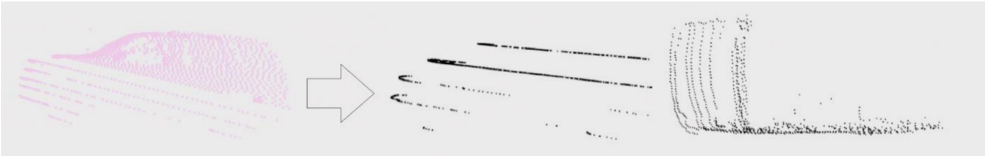

Figure 10. Projection of Pandaset car point cloud to VLP-16 channel distribution.

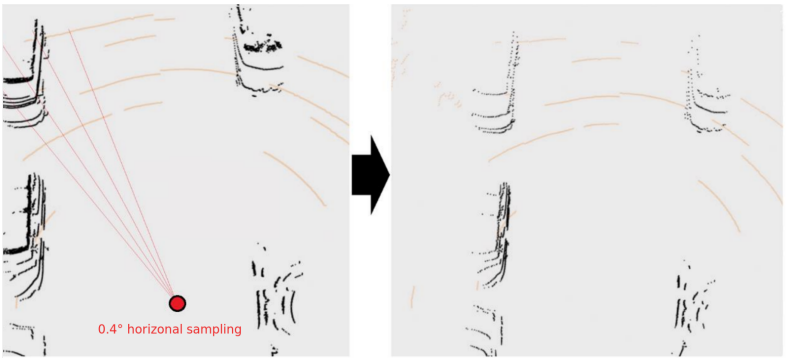

Figure 11. Horizontal scanning of Pandaset scene to match VLP-16 horizontal resolution.

Figures 10 and 11 illustrate the key steps in the conversion workflow applied to an example point cloud from Pandaset. The proposed algorithm adapts object-of-interest point cloud distributions to match the operational parameters of the target

data used in deployment inference. As a result, we introduced two additional testing datasets **Converted KITTI** and **Converted Pandaset**, which reflect adjusted LiDAR's parameters identical with **URset** in terms of Vertical FoV, reduced number of channels operational height and modified horizontal resolution.

Adapting open-source datasets to simulate less feature-rich LiDARs introduces several challenges. The irregular distribution of points in the original datasets can lead to data sparsity after downscaling. Additionally, accurately projecting points onto conical surfaces may introduce errors when the difference in LiDAR's heights is significant, creating distortions in downscaled point clouds. It is important to note that the algorithm only supports downscaling, limiting its applicability when upscaling to a higher number of channels or increasing resolution is required.

Finally, to unify the datasets across different experiments we have adapted the open source datasets also in terms of the number of classes to achieve binary classification. In other words, we have maintained all objects of a class *Car* while all other classes were transformed into a single class *Background*.

4.3. URset_syn

URset_syn was created using the methodology in Section 3, enabling synthetic LiDAR training data generation without manual annotation. Background scans for instance insertions were obtained from robot-recorded sequences in the same parking lot but from different locations than URset as depicted in Figure 9.

Background Scans Selection: A total of n_b=6,740 LiDAR scans without cars were extracted from three different sequences. As in URset, the scan range was limited to 100 meters. These scans were post-processed according to the steps described in Section 3.2.1 to ensure structured scene representation.

Car instance generation: As described in Section 3.1, a dataset of 415 scaled ShapeNet car instances was used. Each car instance was rendered from 12 different viewpoints at varying distances from the LiDAR sensor, ensuring comprehensive object representation across consecutive scans. This process resulted in a dataset of 4980 synthetic scans.

Initially, the maximum instance placement range was set to 20 meters, following a Gaussian distribution ($\mu_d = 10$, $\sigma_d = 20$). However, due to increasing sparsity in 16-channel LiDAR data beyond this range, placement strategies were later refined (Section 5.1).

Object Placement Strategy: Object placement followed the insertion conditions described in Section 3.2.2. The number of cars per scan, denoted as n_i was drawn from a normal distribution with a mean of $\mu_n = 5$ and a standard deviation of $\sigma_n = 4$, sampled randomly from the 4980 synthetic scans.

Each inserted object underwent collision and occlusion verification using ray tracing, with placement adjustments in 10° increments (see Section 3.2.2 for details). Objects lacking sufficient background points in their bounding box (expanded by 10%) were rejected to ensure seamless integration with the environment. If placement failed, a new object was selected outside the initial n_i set. The insertion process

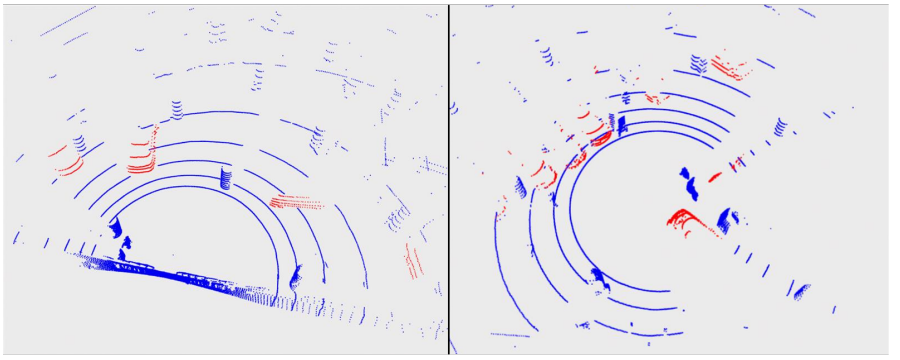

Figure 12. Comparison between an example URset_syn scene (left) and a real-world URset scene (right). Differences in object distribution can be observed, such as the clustering of real-world vehicles in parking areas, whereas synthetic placements are more randomized.

terminated if $k_f = 10$ consecutive objects failed, preventing excessive computation.

Noise Injection and Augmentations: Multiple noise injection strategies were applied to simulate real sensor data imperfections. Optimal values for per-point dropout, Gaussian noise, interior noise, and truncation (occlusion simulation) were detailed in Section 3.3.

An example URset_syn scene compared with URset is presented in Figure 12. Notable differences in object distribution and their clustering can be observed. These differences highlight the challenge of replicating real-world spatial patterns in synthetic data without prior statistical knowledge of their distribution.

5. Experiments

In this section, we will describe three aspects: the influence of various noise augmentation techniques, the comparison with performance of other datasets, and the Ablation Study. For the evaluation, we trained the SphereFormer [18], one of the best-performing LiDAR segmentation networks, on all combinations of datasets. We have decided on a single network to ensure a proper comparison between various datasets. Using other networks would introduce additional complexity to the comparison since we expected that other methods would perform proportionally with various datasets. The metrics used for evaluation are Accuracy and IoU, as suggested by the authors of the Sphereformer. Following training, the evaluation was run for all experiments on the test subset of the **URset**. All discussed experiments were trained from scratch for approximately 20 to 30 epochs, using early stopping to prevent model overfitting.

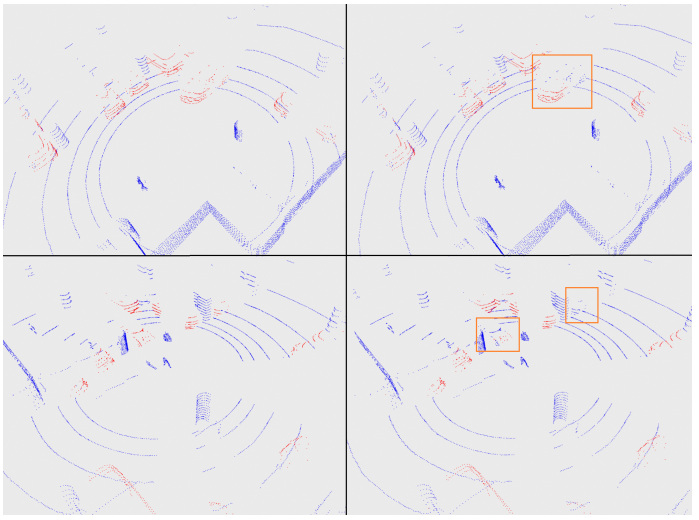

Figure 13. Two examples of a segmented point cloud. Left: URset ground truth data. Right: prediction result of the model trained with URset_syn. Good performance is clearly visible however, the system also fails with small and spatially shrunk clusters.

5.1. Influence of noise injection techniques

Noise injection techniques proposed before have the biggest influence on the results of synthetic datasets, and each of them contributes differently. To clarify to which extent, we performed the verification in a stepwise fashion, sequentially enabling further noise insertion methods. Table 2 summarises the outcomes of various steps. As visible, using synthetic instances without any augmentation leads to mediocre results, while each step improves the metrics gradually. The biggest increase can be spotted after inserting random points in the middle of the car. We attribute this to the fact that synthetic cars are solid mesh models without transparent windows, which prevents the simulation from properly mimicking the interaction with glass, which in reality is transparent and LiDAR sensor penetrates through it to see a car interior. It can also be noticed that our assumption on unprecise manual annotation does not hold as random point class modification does not improve the results. This would suggest class mislabelling is rare. We, therefore skip this technique in further steps. Other techniques have a smaller but noticeable influence.

Finally, after running multiple experiments, we decided to numerically analyse the characteristics of the ground truth dataset to verify what potential aspects are missing when adopting the assumption of pure statistical generation. The simulation framework and insertion algorithms described in Section 3 provide very good data samples so that aspects like point cloud cardinality or numbers of cars per scene are realistic. We noticed, however, that the distribution of car instances along the range differs from the real-life data. Figure 14 shows the distribution of cluster distances

Dropout	Per-point gaussian noise	Random Points Injection	Random points class modification	Cut-off	**Adjusted clusters mean distance**	Accuracy	IoU
-	-	-	-	-	-	68.6	64.7
✓	-	-	-	-	-	70.5	66.3
✓	✓	-	-	-	-	71.6	67.7
✓	✓	✓	-	-	-	75.4	70.3
✓	✓	✓	✓	-	-	75	69.9
✓	✓	✓	-	✓	-	76.2	71.4
✓	✓	✓	-	✓	✓	**77.1**	**73**

Table 2. Results of various noise injection techniques.

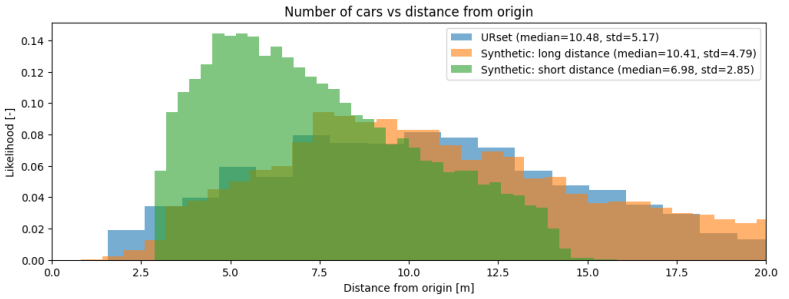

Figure 14. Histogram of cars' distance from the origin for URset (blue), the original distribution of URset_syn (green), and corrected distribution of URset_syn (orange).

between ground truth and synthetic data. The first approach of uniform distribution of car insertion resulted in a skewed distribution. This is the result of insertion conditions, which more often identify available space closer to the robot that usually remains free due to the recording trajectory (the robot was driving in the middle of the driveway). To address this issue, we implemented several strategies, including modifying the Gaussian distribution of synthetic car distances from the LiDAR and introducing a 10% probability of rejecting cars positioned closer than 5 meters during the object placement phase. These adjustments help prevent excessive clustering at short distances, resulting in the distribution shown in Figure 14 (orange). Therefore, in the results Table 2, we added an additional column on *Adjusted clusters mean distance*. One can argue if this information is obvious enough to be deducted just from visual inspection of the ground truth data and does not fall into the assumption of no detailed knowledge of feature distribution.

5.2. Datasets performance comparison

The other set of experiments covered the performance of models trained on various datasets and tested on the test subset of the URset. The results are shown in Table 3. The benchmark results are highlighted by the performance of the train subset (i.e., annotated 70%) of URset, which results in 95.2% Accuracy and 91% of IoU, respectively. This shows that even when trained on the exact same distribution, it is still difficult to reach perfect segmentation. The influence of adaptation of open-source datasets (conversion) is not obvious, and a proper projection strategy is needed as described in Section 4.2. In the case of KITTI, it improves the results while for Pandaset, it deteriorates. We attribute it to the fact that KITTI LiDAR has a better resolution upon the downsampling of channels (as described in Section 4.2), so the resulting point cloud better reflects the actual points. With the lower resolution of Pandar64, the points need to be projected at a bigger distance, which results in unrealistic point cloud modification. Additionally, the altitude in Semantic KITTI is much closer to the URset than in Pandaset.

Training dataset	**Acc**	**IoU**
URset (Ground Truth)	95.2	91.0
S-KITTI	79.5	52.6
Converted S-KITTI	**82.9**	66.7
Pandaset	75.2	70.7
Converted Pandaset	73	65.7
URset_syn	77.1	**73**
URset_syn + S-KITTI + Pandaset	81.3	72.9

Table 3. Results of different datasets tested on the test subset of UR dataset, using Accuracy (Acc) and Intersection over Union (IoU). The top row shows the performance of the manually annotated dataset. Results for URset_syn were obtained with the dataset generated with the best combination of noise augmentation techniques, see Table 2.

What is more, it can be noticed that in general, the open-source datasets perform well in terms of accuracy, but this metric expresses the general performance on all points in the point cloud. Considering, however, that in our use case, only two classes are present (cars and unknown), the IoU metric seems to be more relevant, as it explains how well the specific instances are segmented. Noticeably, the IoU of both these datasets is moderate, and especially in the case of KITTI, the discrepancy between it and the Accuracy is significant. We attribute it to the fact that the car instances in the urban scenario have different orientations to the LiDAR (usually orthogonal or parallel to the sensor) while in the parking scenario, they are more equally distributed, which aligns with the randomness of our generation process. Consequently, the network trained on open-source datasets does not manage to exactly grasp what belongs to a car instance as well as the URset_syn does. Additionally, we believe the density of points per instance of interest also plays a role. One must however notice

the interesting fact that even non-processed datasets show decent results, which suggests the big potential of Transfer Learning. This statement is further supported by the experiment with combined datasets, namely URset_syn + S-KITTI + Pandaset, where an increase in both the Accuracy and the IoU can be noticed compared to datasets treated separately. Finally, our approach performs superior to the tested datasets. Nevertheless, the results are far from the achievements of the ground truth datasets. The exemplary segmented scans are shown in Figure 13.

5.3. Ablation Study

One of the aspects that requires specific attention is the extent of performance degradation with a decreasing amount of data used for training. This is specifically relevant should the system be used for rare occurrences of the objects of interest or limited availability of annotated data. We have conducted multiple experiments to verify how reducing the data influences the performance and at which point synthetically generated datasets bring visible benefits. The results are summarized in Figure 15. Multiple interesting findings emerge from these experiments. Firstly, when the network is trained on a manually annotated dataset originating from the same distribution as the test set, a few hundred scans are enough to achieve decent Accuracy and IoU. What is more, synthetically generated datasets, regardless of the size, struggle to beat manually annotated data. However, there is a breaking point at around 200 scans, below which manually annotated data performs worse in terms of IoU. As this metric is especially relevant in our scenario (focuses on actual instances of cars), the experiments prove that our method is valuable in case of limited availability of the data. Finally, from the plots, it can be noticed that increasing the number of scans in the URset_syn is helpful only to the saturation point (around 7000), after that, the performance drops. We associate it with the fact that with more data, the network fits well to the distribution of the generated dataset and, as a result diverges from the original distribution.

The other set of experiments focused on verifying whether the performance improves when additionally using generated datasets in case of the very limited amount of annotated data. To do so, we run experiments of combined datasets **URset + URset_syn** in various combinations of volumes. We specifically focused on limited sizes of URset to simulate a shortage of data. More specifically, we tested scenarios where we combined 100, 200, and 500 scans of URset, respectively, with various sizes of URset_syn. The results are depicted in Figure 16 in the form of heatmaps. We noticed multiple interesting phenomena. First of all, merging two datasets improves the performance in terms of both the Accuracy and the IoU for multiple cases. It is especially visible for URset sizes of 100 and 200. Adding URset_syn in these cases scores much better than the URset (of this small size) alone. Another observation is that the contribution of both datasets should be equal (i.e. the sizes ratio should remain fifty-fifty). Otherwise, in all cases, increasing or decreasing the URset_syn size deteriorated the result. This also leads to the observation that a big discrepancy leads to significant deterioration, compared to the URset alone, which coincides with

the other experiments and proves that too much data of wrong distribution doesn't help. The biggest benefits of merged datasets are visible for sizes of datasets of 100 and 250 scans each. In the case of 100 scans, the Accuracy increased from 81.4 to 84.1 and Iou from 72.1 to 78.7. Similarly, for 250 scans Accuracy grows from 85.3 to 87.2 and IoU from 74.7 to 79.9.

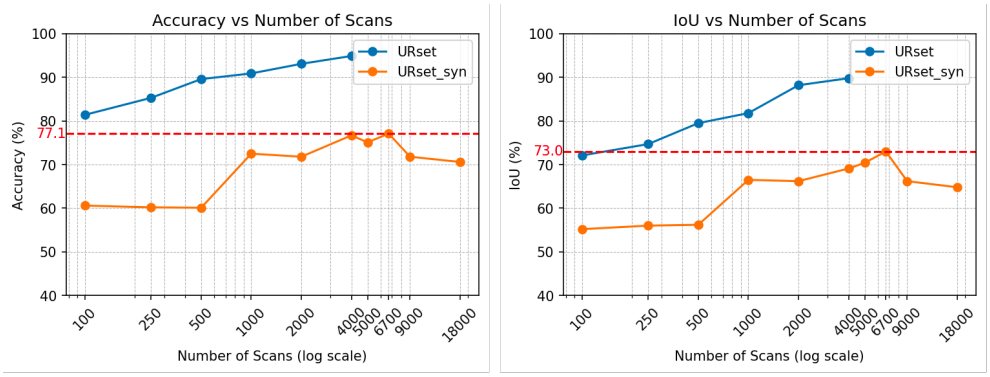

Figure 15. Performance of the segmentation networks for various dataset sizes. The optimal performance of the URset_syn happens between 5000 and 7000 scans, which is marked by a red line. The manually annotated data scores badly only at a few hundred scans.

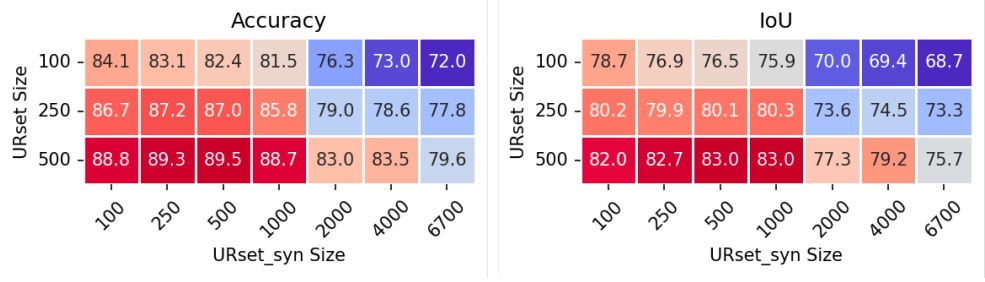

Figure 16. Performance of the system trained on various combinations of URset and URset_syn sizes.

6. Conclusions

We have presented a method to generate training datasets utilizing mixed domains, i.e. recorded background scans populated with synthetically generated instances of objects. Our method results in a good performance of segmentation and allows for quickly generating training datasets with no manual effort, no real-life instances of

objects of interest, and no detailed knowledge about feature distribution in the actual data. What is more, our system introduces no limitation of object types as long as synthetic data corresponds to real objects.

Thanks to the statistical approach and various noise injection techniques, the system can generate datasets that perform superior to the available, open-source datasets without any annotation effort. For the scenarios where the segmentation focuses on the objects of interest only, the system provides a scalable way for adaptation in various environments. We also showed that multiple noise augmentation techniques significantly improve the performance of the segmentation network. The method allows quick creation of training datasets at the cost of accuracy. Thus, it is applicable for a quick setup of a segmentation system in a robot.

Having said that, it is clear that the results of the system perform far from the level of the manually annotated datasets with the identical distribution of features. From the experiments, the direct conclusion is that the annotation remains unbeatable, provided that enough data is available. When this is not the case, our system comes in handy as it can achieve better performance when the data is very limited.

To sum up, we showed that by using purely statistical methods it is possible to generate datasets that exhibit good segmentation, even though they do not achieve the level of actually annotated data. However, the results are of enormous benefit for situations when one faces the absence of proper datasets or a lack of skilled laborers for annotation. We believe achieving results above the ones presented in this paper requires better replication of real-life data (such as data distribution and noise models), which potentially demands methods based on Deep Learning. This ultimately implies that at least some annotated data is needed.

Acknowledgment

The authors would like to thank Jakub Frąszczak for his support with the implementation of the car insertion algorithm. This work was co-financed by the European Union through the European Regional Development Fund under Action 1.1 of the Smart Growth Operational Programme 2014-2020. The project is implemented within the competition organized by the National Centre for Research and Development: Call number: 1/1.1.1/2017 - Industrial research and development work conducted by enterprises. Project Number POIR.01.01.01-00-0206/17: “Designing a prototype of an autonomous platform moving in a production environment”

References

[1] Alonso I., Riazuelo L., Montesano L., and Murillo A. C. Domain adaptation in lidar semantic segmentation by aligning class distributions, 2021.

[2] Arief H. A., Arief M., Zhang G., Liu Z., Bhat M., Indahl U. G., Tveite H., and Zhao D. Sane: smart annotation and evaluation tools for point cloud data. *IEEE*

Access, 8:131848–131858, 2020.

[3] Badrloo S., Varshosaz M., Pirasteh S., and Li J. Image-based obstacle detection methods for the safe navigation of unmanned vehicles: A review. *Remote Sensing*, 14(15):3824, 2022.

[4] Behley J., Garbade M., Milioto A., Quenzel J., Behnke S., Stachniss C., and Gall J. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9297–9307, 2019.

[5] Chang A. X., Funkhouser T., Guibas L., Hanrahan P., Huang Q., Li Z., Savarese S., Savva M., Song S., Su H., et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[6] Choi J., Song Y., and Kwak N. Part-aware data augmentation for 3d object detection in point cloud. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3391–3397. IEEE, 2021.

[7] Cop K., Sułek B., and Trzciński T. Towards efficient generation of data using mixed domains for segmentation of 3d lidar point clouds. In Mańdziuk J., Żychowski A., and Małkiński M., editors, *Progress in Polish Artificial Intelligence Research 5: Proceedings of the 5th Polish Conference on Artificial Intelligence (PP-RAI'2024)*, pages 280–286, Warsaw, Poland, 2024.

[8] CVAT. 3d object annotation. `https://docs.cvat.ai/docs/manual/basics/3d-object-annotation/`. Accessed: 2024-11-28.

[9] Dosovitskiy A., Ros G., Codevilla F., Lopez A., and Koltun V. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[10] Espadinha J., Lebedev I., Lukic L., and Bernardino A. Lidar data noise models and methodology for sim-to-real domain generalization and adaptation in autonomous driving perception. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 797–803. IEEE, 2021.

[11] Fang J., Zhou D., Yan F., Zhao T., Zhang F., Ma Y., Wang L., and Yang R. Augmented lidar simulator for autonomous driving. *IEEE Robotics and Automation Letters*, 5(2):1931–1938, 2020.

[12] Gao B., Pan Y., Li C., Geng S., and Zhao H. Are we hungry for 3d lidar data for semantic segmentation? a survey of datasets and methods. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6063–6081, 2021.

[13] Hesai. Pandar64. `https://www.hesaitech.com/product/pandar64/`. Accessed: 2024-11-28.

[14] Hurl B., Czarnecki K., and Waslander S. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2522–2529. IEEE, 2019.

[15] Ibrahim M., Akhtar N., Wise M., and Mian A. Annotation tool and urban dataset for 3d point cloud semantic segmentation. *IEEE Access*, 9:35984–35996, 2021.

[16] Koenig S. and Howard A. Gazebo: A reliable and versatile robot simulation framework. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 2149–2154. IEEE, 2004.

[17] Kunchev V., Jain L., Ivancevic V., and Finn A. Path planning and obstacle avoidance for autonomous mobile robots: A review. In *Knowledge-Based Intelligent Information and Engineering Systems: 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006. Proceedings, Part II 10*, pages 537–544. Springer, 2006.

[18] Lai X., Chen Y., Lu F., Liu J., and Jia J. Spherical transformer for lidar-based 3d recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17545–17555, 2023.

[19] Manivasagam S., Wang S., Wong K., Zeng W., Sazanovich M., Tan S., Yang B., Ma W.-C., and Urtasun R. Lidarsim: Realistic lidar simulation by leveraging the real world. 2020 ieee. In *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11164–11173, 2020.

[20] Meng Q., Wang W., Zhou T., Shen J., Jia Y., and Van Gool L. Towards a weakly supervised framework for 3d point cloud object detection and annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4454–4468, 2021.

[21] Segments.ai. Data labelling - 3d point cloud. `https://segments.ai/data-labeling/3d-point-cloud/`. Accessed: 2024-11-28.

[22] UnitedRobots. Ur cleaner. `https://unitedrobots.co/ur-cleaner/`. Accessed: 2024-11-28.

[23] Velodyne. Hdl-64e. `https://www.mapix.com/wp-content/uploads/2018/07/63-9194_Rev-J_HDL-64E_S3_Spec-Sheet-Web.pdf`. Accessed: 2024-11-28.

[24] Velodyne. Vlp-16. `https://www.amtechs.co.jp/product/VLP-16-Puck.pdf`.

[25] Wang C., Ma C., Zhu M., and Yang X. Pointaugmenting: Cross-modal augmentation for 3d object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11794–11803, 2021.

[26] Xiao A., Huang J., Guan D., Zhan F., and Lu S. Transfer learning from synthetic to real lidar point cloud for semantic segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2795–2803, 2022.

[27] Xiao P., Shao Z., Hao S., Zhang Z., Chai X., Jiao J., Li Z., Wu J., Sun K., Jiang K., et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101. IEEE, 2021.

[28] Zhang J., Zhao X., Chen Z., and Lu Z. A review of deep learning-based semantic segmentation for point cloud. *IEEE access*, 7:179118–179133, 2019.

Received 01.12.2024, Accepted 17.06.2025