TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

TUM Chair of Geoinformatics, TUM Chair of Photogrammetry and Remote Sensing

Final report of project:

Generating synthetic point clouds of real-world cities for semantic road space segmentation

(Revised version at 9.8.2023)

Authors	Nguyen Duc, Florian Hauck, Yan-Ling Lai, Patrick Madlindl, Xinyuan Zhu
Mentors	M.Sc.M.Sc. Benedikt Schwab, M.Sc. Olaf Wysocki, Prof. Dr. Thomas H. Kolbe
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Aug 2023

Abstract

The generation of synthetic data within simulation environments to partially or fully replace costly and sparse real-world data is important in a diverse range of fields, such as autonomous driving, robotics and medical imaging. In this project, we bring together the domains of geospatial engineering and autonomous driving. We establish how synthetic semantic 3D city models can be used to obtain accurate, simulated laser scan measurements of Light Detection and Ranging (LiDAR) automotive perception sensors.
Our contributions are threefold: (1) We successfully create a pipeline to create synthetic point cloud data based on real-world locations by transferring semantic information from OpenDRIVE and LOD3 models to the synthetic point cloud generation within CARLA, a simulation environment, such that the synthetic point clouds have ground truth semantic information. (2) We further design a novel metric, making use of the Multiscale Model to Model Cloud Comparison (M3C2) and Cloud-to-Cloud (C2C) distances as well as a voxelized Intersection over Union (IoU) approach. This metric enables the quantitative comparison between different synthetic point clouds given their real-world point cloud counterparts. (3) Finally, we explore the domain gap by harnessing the synthetic point clouds for training PointNet++ and KPConv, two semantic segmentation neural networks.

Contents

1 Introduction

The generation of synthetic data within simulation environments to partially or fully replace costly or lacking real-world data is relevant in a diverse range of fields, such as medical imaging, robotics and autonomous driving. In medical imaging, synthetic images of dermatological conditions can complement scant real-world annotated data in training detection networks [8]. Comprehensive simulation environments for various purposes allow for the generation of synthetic data in robotic environments [19]. Recently, there has been active research [6] regarding the creation of synthetic data using simulation and the domain gap between the generated synthetic and real-world-life data. In autonomous driving, robust and provably safe path planning algorithms can be tested and fine-tuned against automatically generated scenarios in simulation before being deployed to real-world test vehicles [13].
As it forms - together with geometric understanding - the backbone of perception and enables scene understanding, *semantic segmentation* is a crucial part of autonomous driving. Today, deep learning-based methods are state-of-the-art in semantic segmentation. While there are some (benchmark) data sets, such as SemanticKITTI [2] and nuScenes [3], the absence of diverse large-scale training data and high cost of collecting and labelling new data pose a challenge. With this project, we put forward a method to create synthetic Light Detection and Ranging (LiDAR) point clouds in the urban road space to enlarge and diversify the available training data.
Three main questions of research drive this project:

1. What are suitable classes and class correspondences for urban environment modeling and road space segmentation using mobile laser scanning (MLS) data?

2. What deterministic and quantitative metric can be developed to compare synthetic point clouds and real-world MLS data?

3. What are the domain disparities and inference performance variations between real-world MLS and synthetic point clouds in tasks such as semantic road space segmentation?

Approach and Contributions

In this project, we bring together the domains of geospatial engineering and autonomous driving by investigating how synthetic city- and building models can be used to obtain accurate, simulated measurements of automotive perception sensors such as LiDAR.
Building data comprising exterior features, such as walls, windows and doors (called Level of Detail (LOD)3) is provided for an area within the city center of Ingolstadt, Germany, by the SAVeNoW project [29, 32]. For the same area, dense MLS as well as OpenDRIVE data is provided by 3D Mapping Solutions [27] within the SAVeNoW project [29].
In order to answer the research questions, we provide four major contributions to the field.

- Firstly, the semantic datasets are analyzed to create a concise yet sufficiently detailed set of classes to describe the urban environment.

- The semantic LOD3 data as well as OpenDRIVE data are combined within CARLA [7], a simulation environment for autonomous driving. This allows for performing virtual LiDAR measurements and creating synthetic point clouds of the inspected area in Ingolstadt.
- The synthetic point clouds obtained by this procedure are expected to be similar to the real-world MLS point clouds in terms of semantics and geometry. To verify this, we directly compare the point clouds by measuring distances and employing a novel comparison metric developed for this project.
- Lastly, we leverage the synthetic data for training segmentation networks using different proportions of real-world and synthetic data and compare inference performance, thereby investigating the domain gap in semantic road space segmentation.

In chapter 2, we briefly introduce related work. Chapters 3 and 4 elaborate on which data has been used in the project and how it has been processed, including the processes of labelling real-world data, generating synthetic data, data comparison and training. Then, in chapter 5, we present and discuss the results of our research. Lastly, we reflect on our work and indicate further research directions in chapter 6.

2 Related Work

Synthetic Point Cloud Dataset The simulators such as CARLA [7] provide the ability to collect simulated LiDAR scans. The deep learning method can use vast data from such environments. However, this method introduces a large domain gap between the real-world and synthetic data, which is still open for research. Datasets such as Paris-CARLA-3D [6] have shown a potential way to create synthetic data conveniently based on 3D models. Moreover, they drive a correlated strategy by directly reconstructing the environment from authentic point clouds using splats [24]. They aim to generate imperfect synthetic scans that resemble real-world point clouds, ultimately reducing the domain gap in training results for semantic segmentation to avoid using inexact geometric models.

Transfer Learning Synthetic Point Cloud Applying knowledge transfer from synthetic to real-world data has been studied widely to mitigate constraints from data annotation in various computer vision tasks, such as semantic segmentation. However, most studies around transfer learning from synthetic to real-world focus on 2D images, while applying synthetic data in 3D point clouds still needs to catch up. Until recently, more research, such as SynLiDAR [34], focuses more on this problem. This method effectively addresses synthetic-to-real-world gaps and consistently enhances point cloud segmentation through its translated data.

TUM FAÇADE Façade segmentation plays a crucial role in 3D semantic segmentation of urban scenes, as buildings serve as one of the most frequent objects in the cities. TUM-FAÇADE[33] pioneers in the evaluation and creation of façade-focused 3D point cloud benchmark for façade segmentation. They propose a method to enrich the existing 3D point cloud benchmark with façad related semantics and introduce TUM-FAÇADE data set. Moreover, they present a standard class list for façade segmentation. The list gives descriptions and definitions of each class and their matching features in CityGML class. In our work, we leverage the list to formulate a consistent class list between MLS point cloud data and CityGML LOD3 building data.

3 Data Preparation

In this chapter, we first elaborate on which data we use, then on labelling the data and finally on how we generate synthetic data and collect synthetic point cloud.

3.1 Data Sets

For this project, data from various sources and of various kinds is being used.

3.1.1 Mobile Laser Scan

The project is provided high-density mobile laser scanning (MLS) data by 3D Mapping Solutions as part of the SAVeNoW project [29] covering parts of the city center of Ingolstadt, Germany. MLS point clouds differ from classical LiDAR scans by their density. While the latter usually have point densities of 50 - 500 points/m^2, MLS data can feature over 5000 points/m^2 [38].

3.1.2 CityGML LOD3 Model

CityGML is an international standard format issued by the Open Geospatial Consortium. It provides a universal style to express and symbolize information from object geometry, semantics, object appearances to objects' hierarchical relationships. It also offers different Levels of Detail (LOD) to portray buildings in the city[10]. With LOD0 the simplest to LOD4 the most intricated, it allows a more flexible depiction of buildings to adapt to various use cases. In our project, we focus our work on LOD3 building models for selected street sections of Ingolstadt[32]. This dataset provided by the SAVeNoW project [29] and the TUM Chair of Geoinformatics contains buildings that consist of the following subfeatures: Door, Window, BuildingPart, BuildingInstallation, WallSurface, RoofSurface, ClosureSurface, GroundSurface, OuterCeilingSurface, and OuterFloorSurface. In the later steps of our project, we leverage these subfeatures to create different segments of 3D meshes that allow the transition of semantics.

3.1.3 OpenDRIVE

OpenDRIVE file [20] provides data regarding the geometry of roads, lanes, and objects, such as roadmarks, as well as road features along the roads, like signals. OpenDRIVE's main purpose is to provide a road network description that can be fed into simulations to develop and validate Advanced Driver Assistant Systems (ADAS) and Automated Driving (AD) features. Combining the LOD3 Model with CityGML dataset, we are able to build the virtual city and later leverage it for generating a synthetic dataset.

3.2 Definition of Classes for the Urban Environment

We define classes for labelling and segmentation in the urban environment and road space based on the existing semantic categories in the CityGML and OpenDRIVE datasets. A high-level goal while designing the class definitions is to keep the numbers of the objects in each class to be roughly identical within one dataset. Thus, we investigate the CityGML

/ LOD3 [32] and OpenDRIVE datasets in the FME Workbench. This provides a variety of information on the properties of the dataset, including the number of occurrences of different CityGML and OpenDRIVE class types. These numbers of occurrences serve as a basis for the definition of our custom high-level classes. In order to obtain class balance as much as possible, we merge similar class types, while also keeping some distinction in important categories, especially within building features. (There, we actively decide to keep features with relatively low occurrence counts, such as doors.)
The result of the merging process is depicted in Appendix Figure 9. The class list is given in Table 1. For concise definitions of the classes, see Appendix Figure 8.

Table 1: Class list for the Urban Environment and road space segmentation.

Class	**Label No.**
`RoadSurface`	1
`GroundSurface`	2
`RoadInstallation`	3
`Vehicle`	4
`Pedestrian`	5
`WallSurface`	6
`RoofSurface`	7
`Door`	8
`Window`	9
`BuildingInstallation`	10
`Tree`	11
`Noise`	12

3.3 Systematic Manual Labelling

To ensure high quality and reliability in the labelling of the MLS data, a labelling process supported by automatic tools has been defined, as well as the specific tools to use.

3.3.1 Labelling Process

The labelling process has been designed to ensure highest possible consistency and quality given the limited resources within the project. It encompasses six stages.

1. Loading in the MLS point cloud into the tool suite used for labelling. Hereby, a transformation from global coordinates to a more handy local coordinate frame can be performed. This transformation is only temporary and will be removed upon saving the manipulated point cloud.

2. Running a connected-component-based geometric classification. This separates points and groups of points that are far away from other points in the region, effectively filtering out outliers and singular objects. The algorithm splits up the original point cloud into N new point clouds, each point cloud containing points that are geometrically close to each other while having some distance to other groups. The indices

(names) of the new point clouds are ordered by the number of points they contain, i.e. the first new point cloud has the highest number of points, while the Nth point cloud has the fewest. With the parameters used, almost all non-noise points are in the first new point cloud, while the other new point clouds contain either some solitary (and still relevant) objects or especially with ascending indices, noise. Thus, the majority of labelling work will be done in the first new point cloud, called from now on for simplicity point cloud.

3. Performing ground plane segmentation. In the point cloud, points belonging to the ground planes, such as roads, pavements and soil, are separated from points belonging to 3D objects with height. This is a crucial step for simplifying and speeding up manual labelling. The ground plane segmentation performs better when separable objects and noise points have previously been removed, which is another justification for step 2 of this process. The ground plane segmentation creates two new point clouds, one containing ground points, the other containing the 3D object points.

4. Manual labelling of the ground point cloud and the non-ground point cloud. This is performed using a polyline segmentation tool. For a selected group of points, the numerical classification value (label) is assigned according to Table 1. This process step is based on a labelling guideline created for this project. The guideline contains the class names, their numerical representation, and a definition of what this class includes and what it does not. The guideline is appended to this report, see Appendix 7 Figure 8. Each point group selected and labelled with the segmentation tool is stored in a new sub-point cloud. This point cloud contains only one class of points and manually has to be given a meaningful name. This is to ensure easy re-labelling at a later time in case of human errors.

5. Merge and export the labelled point cloud. Having labelled all points, the sub-point clouds are duplicated and their duplicates are then merged into one point cloud. That point cloud is then exported as a .las file and contains the labels.

6. Peer-checking of the result. Another member of the project team looks at the labelled point cloud and double-checks the consistency of the labelling. In case of detected errors and inconsistencies, the responsible team member revisits step 4 of this process and adjusts the labelling. The effort of re-labelling is limited as the intermediate results in the form of the sub-point clouds can be used.

3.3.2 Labelling Tools

In this project, the open source software CloudCompare v12 (Kyiv) [5] is used. It contains a connected-component-based geometric labelling tool, a ground plane segmentation tool based on cloth simulation [36] and a manual polyline segmentation tool. With these tools, the labelling process can be efficiently conducted.

3.4 Synthetic Data Pipeline

Our synthetic data pipeline takes in the CityGML model of the LOD3 Building [32] and an OpenDRIVE file that contains the information regarding the road systems in Ingolstadt. The synthetic data pipeline has three main steps: Road Generation, LOD3 Building Generation, and Map Generation.

3.4.1 Road Generation

For the current software to process OpenDRIVE files, we apply two different software, RoadRunner and FME, since each has its advantages and disadvantages. In our project, three different approaches are used to extract these data from the OpenDRIVE file into our synthetic custom map. Figure 1 displays an overview of our dataflow. By applying a different approach, the synthetic custom map generation would be as realistic as possible:

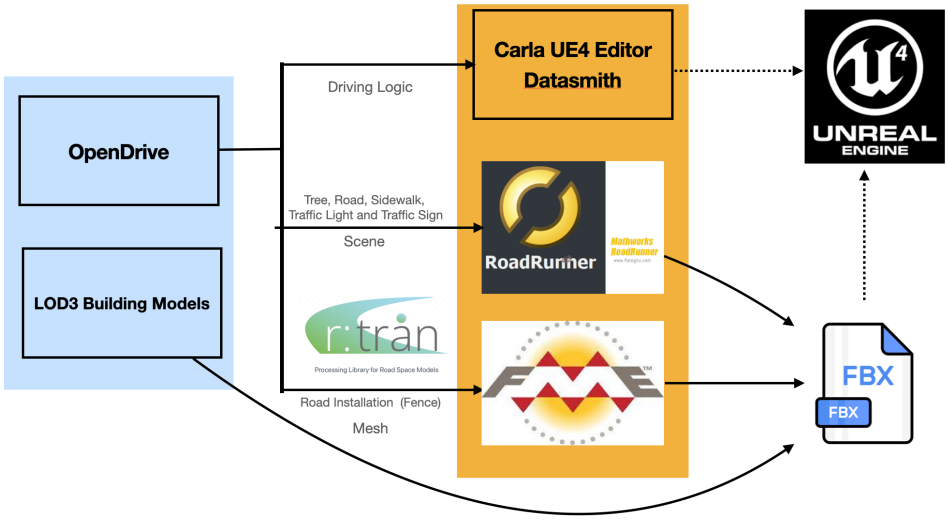


Figure 1: Overview of our dataflow

CARLA [7] is an open-source simulator for autonomous driving research. CARLA allows for simulating LiDAR and camera sensors in virtual outdoor environments. The synthetic LiDAR would be placed on an ego vehicle in CARLA to make the collected point cloud more realistic. The vehicle would drive around the custom synthetic map we created to collect the point cloud data. In the OpenDRIVE file processing step, CARLA is used to interpret the driving logic (laning and road networks) from the OpenDRIVE input file. CARLA would produce spawnable points for vehicles and route planner for the vehicles.

RoadRunner [17] is an interactive editor that allows 3D scene design and road scene customization for the simulation. RoadRunner takes the OpenDRIVE files and produces a scene that contains the fundamental components of the road system in our custom map: Road geometry, the city ground, trees, and some road installations, such as traffic lights

and traffic signs. RoadRunner Scene produces an FBX file containing all the geometry information of the scene in meshes format. This FBX file would be applied later in our pipeline to generate the synthetic map. The mesh geometries of vegetation objects generated by RoadRunner are not accurate compared to real-world data. However, they are positional correct, and their relative measure is also relatively valid, so they are still acceptable when creating our synthetic map.

Feature Manipulation Engine (FME) [25] is a geospatial extract, transformation, and load software platform. FME is applied to generate road installations. FME allows us to customize the mesh transformation from the CityGML definition, which could create better geometry details. FME transformation is more complicated than RoadRunner, especially if we want to keep the semantic matching of the meshes when applied in the CARLA simulator.

r:trån [26] is a road space transformation library. It supports a smooth transformation of road space model from OpenDRIVE to CityGML format, with the features in OpenDRIVE preserved nicely. Since FME does not support OpenDRIVE format, we utilized r:trån to transform our OpenDRIVE data into CityGML format in order to do further transformations on the data.

3.4.2 LOD3 Building Generation

To create custom 3D buildings, we need to transform CityGML file into CARLA-readable format which is a FBX mesh file. We leverage FME to do the transformation: First, we add a reader that reads in CityGML with individual feature type. The choice of reading the features as individuals is important, since it allows us to produce separate mesh files for each feature. By doing so, we can keep the object semantics in our custom map for CARLA. After reading in the CityGML files with individual features, we pass each subfeature of our LOD3 models [32] through the following transformers in listing order:

1. GeometryFilter This filter can be used to filter out geometry types that are specified. We use it to filter out geometry type "surface", and pass them to later stages.

2. GeometryValidator This filter is used to detect invalid surfaces and if possible, fix the surfaces. For example, we detect and repair surfaces with boundaries that look connected when projected to X-Y plane, but are actually not well connected on Z coordinate.

3. Triangulator We use the Triangulator transformer to take in surfaces and triangulate them into meshes.

4. MeshMerger We then use MeshMerger to combine the triangle meshes we have for each surface into one mesh.

5. Offsetter We use Offsetter to translate the coordinates of the meshes in order to align with the road space model.
After transformation, we have for each feature type a single merged mesh. To save them, we add a data writer to write the output meshes into FBX mesh files. We let the writer copy the feature types from the reader so that it creates for each of them an output flow.

3.4.3 Road Installation

Road installations, such as fences, poles, and traffic lights, exist widely in the real-world. To simulate them, we extract the feature *CityFurniture* from the road space model. We first utilize r:trån to transform OpenDRIVE data into CityGML model. We then leverage FME to extract the CityFurniture objects and transform them into a mesh file. Again, we use Triangulator, MeshMerger, and Offsetter to do the transformation. The detailed visialization can be found in Appendix 11.

3.4.4 Map Generation

CARLA uses Unreal Engine Editor 4 to create and modify its maps. CARLA Unreal Engine Editor 4 (CARLAUE4) contains all the assets and scenes for generating the CARLA binary. The CARLAUE4 contains all the functionality of CARLA and is kept separate from most of the assets, so the functionality in this plugin can be used as much as possible in any Unreal project.

Meshes Alignment in map generation In the CARLAUE4, we receive three FBX files from the inputs: OpenDRIVE-RoadRunner meshes (comprising the classes road, sidewalk, trees, bushes, ground, traffic light, and traffic signs), LOD3 building model meshes, and one OpenDRIVE-FME mesh (containing road installation features). Furthermore, CARLAUE4 would take in the OpenDRIVE file directly and produces the road logic, which contains spawnable points and a route planner. These FBX files, when imported into Unreal Engine, have lost information regarding the global position and hence need to align together. The LOD3 building was exported in the wrong scale (meter) while the other was scaled to centimeters; hence, the LOD3 building models needs to be scaled 100 times in all dimensions. The rotation is maintained, so it is 3 Degrees of Freedom (DoF) alignment corresponding to 3 coordinates. The meshes can be aligned by choosing a pair of points that are matching to each other and fit them accordingly.

Semantic matching Instead of using the semantics given by CARLA, custom semantic tags in CARLA were created and used corresponding to our classes (see Table 1) for real-world data. This step is based on CARLA's guideline documentation to update some of the CARLA source code. Then each mesh would be matched to one semantic tag by sorting the assets in Unreal Engine accordingly.

3.5 Synthetic Data Generation

After creating our synthetic data pipelines, we use a manually controlled vehicle in CARLA to make the collected synthetic point cloud more realistic. The synthetic point cloud is collected by two sensors on top of our vehicle.

LiDAR Sensor configuration Our simulated LiDAR sensors' configuration is based on a real-life LiDAR setup. Based on the real-life configuration supplied by [11] and [6], we update our simulated LiDAR setup accordingly. The details of our LiDAR setup are provided in Appendix 7.

Manual Control Vehicle Setup In order to make the driving setup more realistic and avoid the numeric complexity of designing a driving path, our ego vehicle is controlled manually. Based on the script given by CARLA, we added more functionality to collect and align the point cloud on the run and the option to save the point cloud. While collecting the point cloud, the ego vehicle speed is maintained at around 30km/h. To avoid physical simulation problems (such as invisible bounding box collision), most of the physical interactions of the road and building meshes are turned off, and only sensor interaction is allowed. The vehicle is driven based on the road logic. Furthermore, the ego vehicle only acts as our sensors carrier, and hence a small deviation from the meshes and driving logic, which could lead to non-perfect alignment between the vehicles and road, has no notable effect on the experiment outcome. With this approach, the synthetic point cloud can be collected quickly and conveniently. The complete implementation can be found in our repository [18].

Noise Injection In CARLA, noise for synthetic LiDAR sensors is not yet supported. Thus, before registering the synthetic LiDAR scans, we post-process the individual point clouds by injecting noise. The process is depicted in Figure 2. The semantic LiDAR scanner in CARLA collects points in cartesian coordinates and converts them to spherical coordinates. Here, the error in range values is normally distributed with a standard deviation of $\rho = 2$ cm, which produces promising results based on the work of [28]. After applying the noise to the point cloud in spherical coordinates, the point cloud is converted back to cartesian coordinates. Finally, we apply rotation and translation matrix based on the current sensors' position for alignment.

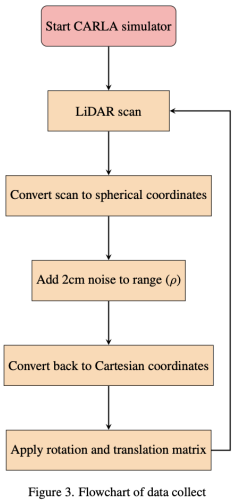


Figure 2: Carla noise injection process [28]

4 Data Processing

One major goal of this project is to evaluate the usefulness and correctness of the generated synthetic data. To this end, an indirect and a direct approach are taken. The direct approach involves computing distances and designing a custom metric to quantify how well the synthetic point clouds match the real-world ones. As an indirect approach, we test the synthetic point cloud by application. We train semantic segmentation deep learning networks on synthetic data, real-world data, and on different mixtures of these. Then, by comparing inference measures, we aim to assess how suitable the synthetic data is for training. In both approaches, the labelled real-world MLS data is necessary.

4.1 Direct Comparison: Distances and Metric

In order to compare the synthetic point cloud with the labelled MLS point cloud and measure similarity, different distances can be employed. For reproducibility, we create a Python script that conducts the distance- and metric calculation. For the benefit of the community, we publish the script in [16].
Before comparison, it is necessary to transform the two point clouds into the same coordinate frame. This step can be conducted and visually verified in CloudCompare [5]. Another preprocessing step is to conduct a top-view grid filtering (see also Appendix 7 Figure 15). This rough approach ensures that in the area of comparison, both point clouds have at least some points. For the comparison script, we implemented the transformations and filtering directly in the code.

4.1.1 State-of-the-Art Metrics for Point Cloud Comparison

There are several distances and metrics that can be computed between two sets $\mathcal{A}$ and $\mathcal{B}$. In semantic segmentation, the Jaccard index $J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}$, also known as intersection over union (IoU), is widely used. In semantic segmentation, it can indicate class-wise segmentation performance. Calculating the IoU for each class individually and taking the average results in the mean intersection over union (mIoU).
While the IoU is useful in semantic evaluations, for geometric comparisons, different distances are preferred. The Hausdorff-Distance encodes the longest shortest distance between two elements of $\mathcal{A}$ and $\mathcal{B}$. In other words, it is the largest distance between the convex hulls of the two sets. Practically, this means that outliers affect the Hausdorff distance.
If point-wise correspondences between $\mathcal{A}$ and $\mathcal{B}$ are known, a point-to-point distance can be computed by calculating the mean or median of the distances between the correspondences. If there are no correspondences given, an alternative approach is for every $a_i \in \mathcal{A}$ to compute (1) the distance to the nearest neighbor-point in $\mathcal{B}$ or (2) the distance to a mesh generated by the points of $\mathcal{B}$. In this setup, $\mathcal{B}$ is called a reference point cloud, and $\mathcal{A}$ is called the compared point cloud. For a valid result, the reference point cloud should overlap the compared or target point cloud. Subsequently, we will call these approaches cloud-to-cloud distance (1) and cloud-to-mesh distance (2) respectively [9, 12].
In the field of geodesy, change detection algorithms are widely used to monitor changes in topography, vegetation, or also cities. One popular classical approach is multi scale model

to model cloud comparison (M3C2) [12, 14]. M3C2 first estimates surface normals and orientations and secondly estimates a mean change of surface along the normal direction [12]. The result of the algorithm is a new point cloud $\mathbf{d}_{M3C2}$, in which points encode the respective M3C2 distance (or change) at their position.
While M3C2 is more sophisticated than C2C, a limitation is that change is measured along the directions of the surface normals. For this, see also Appendix, Figure 17. Thus, changes in a direction parallel to surfaces are, as long as there are no correspondences given, difficult to estimate. This is also true for C2C and C2M [12]. An extension of M3C2 addressing this issue is Correspondence-Driven Plane-Based M3C2 [12, 35]. Next to classical methods, there are also deep learning based change detection approaches [12]. These require training data sets [12].

4.1.2 Our Approach

To compare real-world and synthetic point clouds, for this project, we decide to compute (1) a distance between the point clouds and (2) create a custom metric that shall ease the comparison and benchmarking between point clouds.
Due to the lack of correspondences and change detection training data, the selection of distances and metrics is limited to Hausdorff distance, Jaccard index, C2C, C2M and M3C2.

Distance Calculation Experiments with the Hausdorff distance have led to the insight that it is expensive to compute and affected by outliers. As we want to compare the point clouds with only simple preprocessing methods described above, there will still be outlier objects. These render the Hausdorff distance ineffective for our means, as outliers with a distance of up to the filter-grid size are still in the point clouds. For instance, if there is a car at the very border of $\mathcal{A}$ and point cloud $\mathcal{B}$ does not cover that area, the distance from the points constituting the car to the nearest points in $\mathcal{B}$ is high. As only the highest such distance is represented in the Hausdorff distance, we get no information on how the rest of the clouds match (except for that they are closer than the outlier).[1]
This is why we chose to compute a C2C distance d_{C2C} between the two point clouds. Here, the outliers still have an impact on the result, but the much more numerous inliers will dominate - thus, the result is more meaningful for an indication of how well the synthetic point cloud matches geometrically with the real-world one.
Additionally, we compute the M3C2 distance. To make use of the semantic information that we have, we perform the M3C2 calculation class-wise and compute a weighted average. Let C be the number of classes in both $\mathcal{A}$ and $\mathcal{B}$. Then we split both point clouds into disjoint class-wise point clouds, i.e. $\mathcal{A} = \bigcup_{c=1}^{C} \mathcal{A}^{(c)}$ and $\mathcal{B} = \bigcup_{c=1}^{C} \mathcal{B}^{(c)}$. On each pair of class-wise point clouds, we perform M3C2 and take the median of the resulting M3C2 distances. We denote the class wise median as $\widetilde{\mathbf{d}}^{(\mathbf{c})}_{\mathbf{M3C2}}$. Averaging the absolute values of the medians with custom class weights $W = \{w^{(c)}\}_{c=1}^{C}$, we arrive at the mean M3C2

[1] In fact, we implemented and calculated the Hausdorff distance. For one early synthetic point cloud and the corresponding real-world point cloud, the calculation took 5 hours on a modern notebook and the result without grid-filtering was a distance of 75 m.

distance:

$$d_{MeanM3C2} = \sum_{c=1}^{C} w^{(c)} \cdot |\widetilde{\mathbf{d}}^{(\mathbf{c})}_{\mathbf{M3C2}}| \qquad \text{where } \sum_{c=1}^{C} w^{(c)} = 1. \tag{1}$$

The final cloud distance we propose is a weighted mean between the absolute value of C2C distance $|d_{C2C}|$ and the mean M3C2 distance $d_{MeanM3C2}$:

$$d(\mathcal{A}, \mathcal{B}) = \lambda_1 d_{MeanM3C2} + \lambda_2 |d_{C2C}| \qquad \text{where } \lambda_1 + \lambda_2 = 1. \tag{2}$$

Note that the distance is in the same unit as point clouds $\mathcal{A}$ and $\mathcal{B}$ and thus has physical meaning.

Metric Definition For the metric, we incorporate M3C2, C2C and mIoU and fit the result to the interval $[0, 1]$, where 0 is the best possible result, and where 1 means the point clouds are as different as possible. For the fitting, we use a bounded growth function with growth parameter $\alpha < 0 : f(x) = 1 - \texttt{exp}(\alpha x)$.
To compute IoU on 3D data efficiently, we create one voxel grid that we populate with points from both class-wise point clouds. The voxels have a side length of l_{voxel}. The union count is the number of voxels in which there are points from at least one cloud; the intersection count is the number of voxels in which there are points from both clouds. For the mIoU, we weigh the class-wise $IoU^{(c)}$ values with the custom weights W. Thus we have

$$mIoU = \sum_{c=1}^{C} w^{(c)} \cdot IoU^{(c)} \qquad \text{where } \sum_{c=1}^{C} w^{(c)} = 1. \tag{3}$$

The mIoU behaves inversely in the sense that beneficial mIoU values are close to 1 and poor values close to 0. This is why we define a mIoU factor $f_{mIoU} = \frac{1}{mIoU+\varepsilon}$, where $\varepsilon > 0$ is a small value to prevent division by zero.
In the final metric we weigh and insert the distances $|d_{C2C}|$ and $d_{MeanM3C2}$ and the mIoU factor f_{mIoU} into the bounded growth function:

$$m(\mathcal{A}, \mathcal{B}) = 1 - \texttt{exp}\Big(\alpha\big(\lambda_1 d_{MeanM3C2} + \lambda_2 |d_{C2C}| + \lambda_3 f_{mIoU}\big)\Big) \text{ where } \lambda_1 + \lambda_2 + \lambda_3 = 1. \tag{4}$$

Note that, in contrast to the distance (equation 2), the interpretation of the absolute value of this metric is difficult. Instead, the metric is useful when comparing the similarities of different synthetic point clouds to their respective twin real-world point clouds.

Implementation The Python script [16] we implemented for the calculation of distances and metrics uses py4dgeo [21] for the M3C2 calculations, open3d [37] for C2C and laspy [15] for reading the point clouds. All other functionality is implemented by the project team using standard Python libraries.

4.2 Indirect Comparison: Network Training with Synthetic Data

From robotics to architecture, 3D semantic segmentation networks are widely used in various fields. They are especially crucial in the area of autonomous driving, as their capability to understand 3D outdoor scenes in detail facilitates the development of safe and reliable navigation and decision makings. To train a 3D semantic segmentation network that achieves desirable accuracy, the amount and quality of data are of great importance. In reality, the acquisition of outdoor point clouds is not only time-consuming and costly, but also poses the problems of occlusions and missing parts of buildings. This raises the following question: Instead of spending more time and putting more cost into collecting new and high-quality data, can one leverage synthetic point clouds to mend the data shortage and effectively train the neural network? To answer this question, we select two semantic segmentation networks and evaluate the feasibility to train them with our synthetic data.

4.2.1 State-of-the-Art Deep Neural Networks for Semantic Segmentation

We choose two networks that are widely used in 3D segmentation problems: PointNet++ [23] and KPConv [30] for comparison. While they both leverage spatial property of the point clouds, the former one integrates the geospatial information with multi-layer-perceptrons stacking up in a hierarchical structure and the later one leverages 3D kernels to aggregate the information.

PointNet++ [23] PointNet++, as a refinement over the original PointNet[22] architecture, significantly enhances the treatment of unstructured 3D point cloud data by incorporating a hierarchical learning mechanism. The essence of this enhancement stems from overcoming a primary limitation of the original PointNet architecture—its inability to adequately capture local features and multi-scale data. The core idea behind PointNet++ is a nested application of PointNet—deemed "small nets"—across hierarchical levels within the point cloud. The hierarchical strategy deployed in PointNet++ enables the learning of local features with progressively increasing receptive fields, giving rise to a more contextual understanding of the data. In the context of PointNet++'s performance in point cloud segmentation, the effectiveness of the hierarchical approach is evident. By methodically attending to localized and global features in a systematic, hierarchical manner, PointNet++ offers an enhanced framework for point cloud analysis.

KPConv [30] Kernel Point Cloud Convolution is a state-of-the-art convolution network-based approach for unordered point cloud classification and segmentation. Different from other non-convolution works, it approaches the understanding of unordered point cloud by comprehending their local spatial structure. Instead of defining kernels with pixels or voxels like other similar approaches in convolution-based network, they use 3D kernel points to describe the locations of kernel weights. Kernel points allow the kernels to operate directly on point clouds, and thus eliminate the need to project the point clouds onto 2D plane or 3D grids. Moreover, they operate the kernels on spherical neighborhoods. This gives KPConv several advantages over other kernel-point-based methods. Compared to computing convolutions on the whole point cloud, it largely lower the computation

burden. The spherical shape also provides a more consistent domain compared to other methods that find the kernel domain using k-nearest neighbor. Overall, KPConv is an ideal network for analyzing 3D point clouds with their spatial distributions.

4.2.2 PointNet++

Our initial thought is to mitigate any potential distortions in performance outcomes due to the difference in implementations, so we utilize the original repository from Qi et al.[22] for training. But after several attempts, we decided to only use the structure of PointNet++ and wrote other parts of the neural network by ourselves. The main reasons are as follows: (1) The original repository is not maintained for a long time, since the inception of the architecture, with the paper published as far back as 2017. During the implementation period, many of the utilized packages, such as Python, other Python libraries, CUDA have released newer versions with modified functions and features. Therefore, updating the original repository to align with the more recent versions of these packages would necessitate extensive modifications to the codebase. (2) PointNet++ was tested on several benchmark datasets, including ModelNet40[31], ShapeNet[4], and Stanford 3D Indoor Spaces Dataset (S3DIS)[1], and then they were stored in HDF5 format. After being converted into HDF5 format, the dataset is structured into a hierarchical form consisting all individual 3D models or scenes in point cloud representation. Since our dataset is an entire huge outdoor scenario which represents one part of Ingolstadt city, it will take lots of effort if we manually split the dataset into small parts/scenarios in a similar form of the dataset mentioned above. And it will have a bad influence on the performance of the model as well, which we will explain later. To sum up, considering the effort required and the uncertainty (we might face some issues but cannot get feedback from the author since its not maintained anymore) to successfully run PointNet++ from the original repository, we have decided to build the neural network by ourselves.

Data Preparation After using the same preprocessing methods that are applied in KPConv, which generate compressed, downsampled point spheres and not achieving pleasing results with PointNet++ as, for instance the point sphere did not guarantee a consistent size, we approached the data preprocessing for PointNet++ in another way:
The data sets were split by applying a grid and segmenting it into cubes of size 6x6x6m along the raster. Since PointNet++, like many other deep learning models, expects a fixed input size for each batch, the size of the preprocessed data splits was fixed after some manual feature inspections to 12k points. To not mistakenly cut objects in half and therefore losing information about the point interaction in global space by slicing the objects closed shape as well as to also enlarge the data available for training, the sliding cube area was shifted by half its size, in other words by 3m in every direction. Furthermore, every training sample was randomly rotated around the z-axis to augment the cubes a bit. We did not want to apply any artificial noise as the real-world data is already slightly imperfect by nature.

Training on Real World Data When it comes to training, one must take care of the imbalanced distribution of the occurrence of each class. The imbalance can be exemplary seen in 14. `RoadSurface`, `GroundSurface` and `WallSurface` heavily outweigh the other

classes with a share of around 90% of the points in the provided real-world data set. After preprocessing, the share shifts slightly towards walls and trees due to the way of the method of the sliding downsampling window. The previously by density-dominating road surfaces are reduced much more than the spaced-out crowns of partially freestanding trees. Therefore a shift is noticeable and recorded in the appended Figure 18. Nevertheless, the imbalance of class distribution remains, although different, still existent. After all, the standard unweighted multiclass cross-entropy loss approach will therefore not lead to pleasing results. The loss function(s) should be picked with respect to the class imbalance. Therefore a linear combination of Focal Loss and Dice Loss was chosen.

The Focal Loss is given by:

$$\texttt{Focal Loss} = -\alpha \cdot (1-p)^{\gamma} \cdot \log(p) \tag{5}$$

Where p is the predicted probability of the positive class, α is the balancing factor that can be used to adjust the weight of the positive class relative to the negative class. This is used to inversely apply the class balance proportion in order to give more impact to less represented classes on the loss function. It generally encourages the model to down-weight easy examples that are highly represented and focus the training on hard negatives. This purpose is fulfilled by γ, the focusing parameter that adjusts the rate at which easy examples are down-weighted. When $\gamma = 0$, the focal loss reduces to the standard (weighted) cross-entropy loss.

The Dice coefficient is given by:

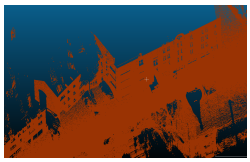
$$\texttt{Dice Coeff.} = \frac{2 \cdot |y \cap \hat{y}|}{|y| + |\hat{y}|} \tag{6}$$

Dice Loss was added as an extension to Focal Loss to improve the IoU performance. The measurement of overlap between the predicted labels and ground truth ranging from 0 to 1, with 1 representing complete overlap, is hereby taken specifically into consideration by the loss function.

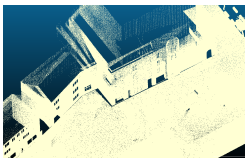
The results of the real-world training on PointNet++ can be found in 5.3. Training was performed on real-world data only since we found KPConv to deliver more pleasing results even after modifications to the PointNet++ structure, loss function and choice of different hyperparameters. Therefore we chose to extend the tests on compositions of synthetic and real-world data on the KPConv network which is introduced in the following.

4.2.3 KPConv

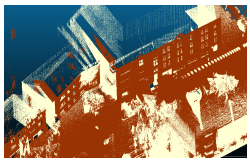
To minimize the effect of performance biases caused by implementation differences, we utilize the original repository from Thomas et al.[30] for training. We do not do any extra hyperparameter tuning in order to keep our experiments fully focus on the effect of training with different data sets. Their program processes large point clouds directly without the need to split the data into small scenes beforehand. This highly matches with our case since our data are large point cloud files with more than millions of points contained. The point clouds are processed and formulated into input batches with a unique method, which contains the following steps:



(a) real-world points



(b) synthetic points



(c) mixture

Figure 3: Mixture of points with 75% real-world and 25% synthetics.

Grid Subsampling This is to down-sample the point clouds in a grid fashion. Point clouds are first being divided into small grids, then one point per grid is kept. The down-sampling step allows less computational burden and memory consumption.

KDTree Construction For each point cloud, a kd-tree is constructed for sphere neighborhood querying around selected center points.

Batch Formulation In each iteration, several random points are selected. Then the spherical neighborhoods of these points are queried and re-centered to the origin. These neighborhoods are then stacked together as an input batch to the network. To formulate input batches that sample different regions of the clouds evenly, a potential value for each point is maintained. Upon selection, the potential value of the point and its neighbors will be updated to a higher value and thus lowering the probability for them to be selected in later batches.
To evaluate the effects of synthetic point clouds, we train the network with the following combinations of real-world and synthetic data: 100% real-world data, 75% real-world data and 25% synthetic data, 50% real-world data and 50% of synthetic data, 25% real-world data and 75% synthetic data, and 100% of synthetic data. The main goal is to see how the network performs when we gradually increase the synthetic data and decrease the real-world data in training. We mix the real-world and synthetic points by first randomly down-sampling the real-world and synthetic point clouds to desired percentage, then concatenate the points into one point cloud. The Figures 5, 6, and 7 visualize the result for a 50-50 mixture of real-world and synthetic data. One can observe the synthetic point cloud mend up a region of wall surface in real-world point cloud that is occluded by tree.
We are also interested in exploring other use cases of synthetic point clouds. For example, can they serve as extra structures for the model to learn more descriptive features to adapt the domains between two kinds of data? Or can they serve as data for pre-training the model, and allow us to do efficient transfer learning on real-world data? To evaluate this, we designed two more experiments as follows: 1. Train on all real-world training data and all synthetic training data, and evaluate on real-world validation data. 2. Train on all synthetic training data for several epochs, then train on all real-world train data, and evaluate on real-world validation data.

5 Results and Discussion

In this section, we present and discuss the results of our experiments. We provide analyses to the research questions of how to quantitatively compare the real-world with synthetic data and investigate the domain gap of synthetic data when using it for semantic road space segmentation. Also, we present the resulting synthetic point clouds generated from the data generation pipeline.

5.1 Generated Synthetic Point Cloud

After applying the data pipelines and generation in 3.4 and 5.1, the final synthetic point cloud contains 100M points. The semantic classes correspond to the real-world data classes. The Figure 4 display the complete synthetic point cloud and the comparison between synthetic point cloud and real-world point cloud. As mentioned in 3.4, vegetation in our map is not geometrically accurate, which is displayed in Figure 5.

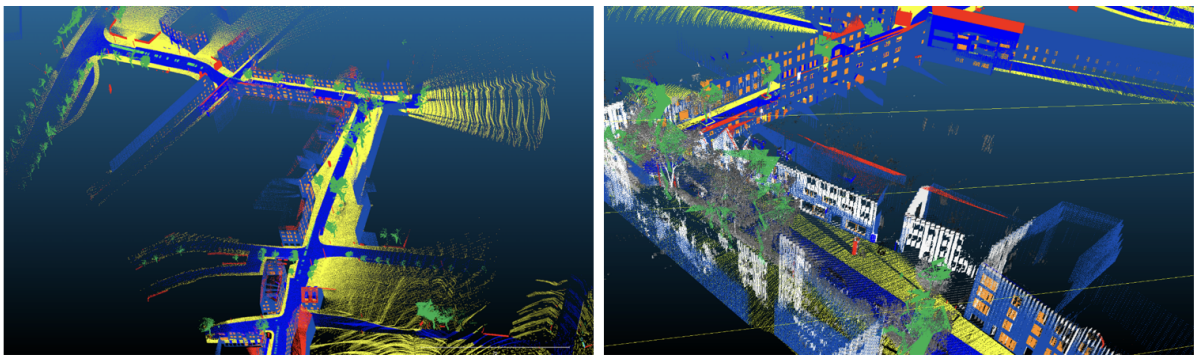


Figure 4: Complete synthetic point cloud (color) and comparison with real data (grey)

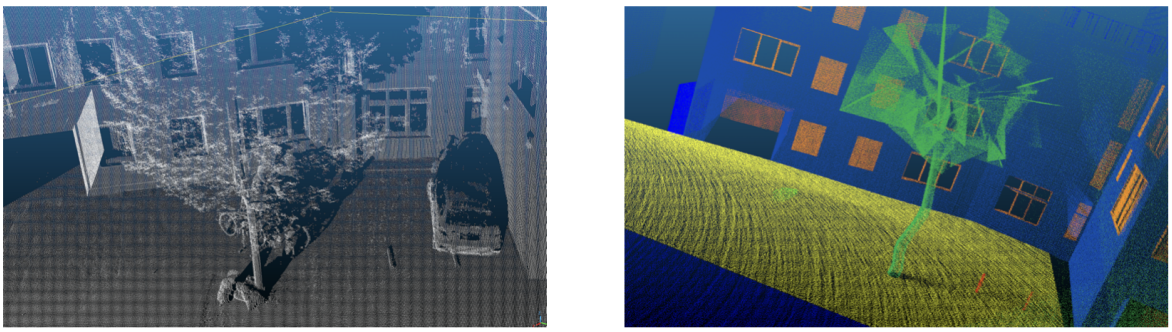


Figure 5: Tree in real (left) and synthetic data (right)

5.2 Cloud Comparison

Parameters For M3C2, we choose parameters proposed by CloudCompare's M3C2 plugin for our data, see Appendix 7.3. For mIoU, we chose a grid size of 5 m for filtering

and a voxel size of $l_{voxel} = 0.5$ m. The transformation offsets to bring both point clouds to the same reference frame are given in Table 2. For the synthetic point cloud, we found some systematic alignment errors and have a second set of (optional) transformations, in the following called *synthetic aligned*. Additionally, the synthetic point cloud needs to be flipped along Y. We suspect this originates in inconsistent use of left-handed or right-handed coordinate systems in the simulation pipeline.

Table 2: Offsets for real-world and synthetic point clouds.

	real-world	Synthetic	Synthetic (Aligned)
X	-674000	0	-0.3
Y	-5405000	0	-1.4
Z	0	0	-1.8

The class weights for mIoU and mean M3C2 are chosen to emphasize building-related classes and can be found in Table 3. We consider walls to be especially important as they constitute the main geometry of buildings. Note that we do not perform the comparison on the classes `Vehicle`, `Pedestrian`, `RoadInstallation`, `Tree` and `Noise`, as these classes are highly temporary or noisy.

Table 3: Weights per class used for weighted average computation mIoU and M3C2

Class	`RoadInst.`	`GroundInst.`	`WallSurf.`	`RoofSurf.`	`Door`	`Window`	`BuildingInst.`
Weight	0.1	0.1	0.2	0.15	0.15	0.15	0.15

As distance weights we use $\lambda_1 = 0.6$ for mean M3C2 and $\lambda_2 = 0.4$ for C2C, assigning more importance to the class-wise change detection than to the outlier-influenced cloud distance. For the metric, we have a similar strategy in choosing the weights and consider the fact that by construction, even acceptable mIoU values will lead to large IoU factors. To have a balance between mean M3C2 and mIoU, we thus empirically select $\lambda_1 = 0.6$ for M3C2 and $\lambda_3 = 0.1$ for mIoU. Again, we attribute less influence to C2C with $\lambda_2 = 0.3$. For the bounded growth rate, we empirically choose $\alpha = -0,2$ as it leads to a wide range of values in the range $[0, 1]$ being used with the data for this project.

Results and Findings For testing the distance and metric calculation, we select a subset of the labelled training dataset called Train2. We use our implemented Python script [16] and the parameters above. The results are in Table 4.

Table 4: Distance and metric results

	distance	**metric**	$\mathbf{d_{MeanM3C2}}$	$\mathbf{d_{C2C}}$	**mIoU**	$\mathbf{f_{mIoU}}$
Synthetic	1.14	0.73	1.01	1.34	1.82%	54.79
Synthetic (Aligned)	0.15	0.09	0.04	0.31	29.01%	3.44

With the alignment shift of the synthetic point cloud, the distances are in a cm range, while the mIoU is in a low two-digit span. This leads to a relatively small IoU factor of 3.44. Accordingly, the aligned synthetic point cloud is performing well on our custom metric with a value of 0.09, which is much closer to 0 than to 1.

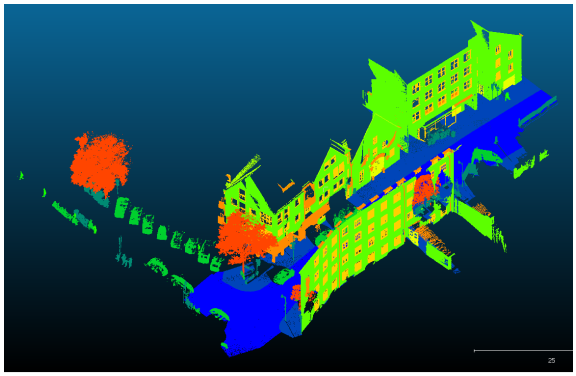
For the non-aligned synthetic point cloud, the distances are at around 1 m, which coincides well with the magnitude of the alignment shift (see Table 2). The mIoU has worsened by one order of magnitude. This is explainable with the voxel size $l_{voxel} = 0.5 \text{ m} < 1.14 = d$ being smaller than the cloud distance. Essentially, the resolution of the IoU calculation is higher than the distance of points, which leads to low IoU results. In contrast, with the aligned synthetic point cloud $d = 0.15 < 0.5 = l_{voxel}$, thus the IoU is significantly higher. The low mIoU as well as the higher distances lead to a metric score of 0.73, which is closer to 1 than to 0 and much higher than the shifted score of 0.09. As the shifted point cloud fits the real-world one better than the unshifted one, this result confirms that our metric works as intended and can indeed distinguish well-fitting semantic point clouds from less-well-fitting ones.

5.3 Training / Using Synthetic Data for Semantic Segmentation

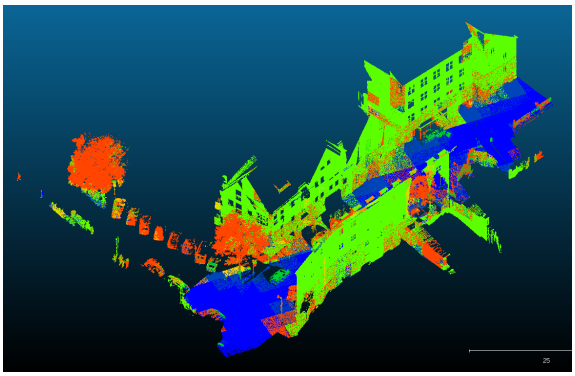
5.3.1 PointNet++

The evaluation of the results when applying the model to the real-world data validation split leads to the following segmentation as shown in Figure 6.
One can see the labelled classes and, especially important, in Figure 7(c), the blue areas, which indicate mislabelled points. Therefore, we can discern from the graph that a considerable percentage of points have not been accurately labeled.
Furthermore, the confusion matrix for this evaluation is shown in Table 8, where the mismatches of the predicted point labels to the ground truth can be analysed in more detail. The most well-segmented classes are `RoadSurface`, `WallSurface`, `Tree`, `GroundSurface`, and `Vehicle` as can be observed in Figure 6, which the confusion matrix in Table 8 confirms by a relatively low False-Negative-Rate and a better performing Precision. Nevertheless, the `Vehicle` class has many wrongly segmented cars in the left area of the view in Figure 6. This area should probably have been cut off in any case since this area was hardly covered by the LiDAR sensors with which the data was recorded.

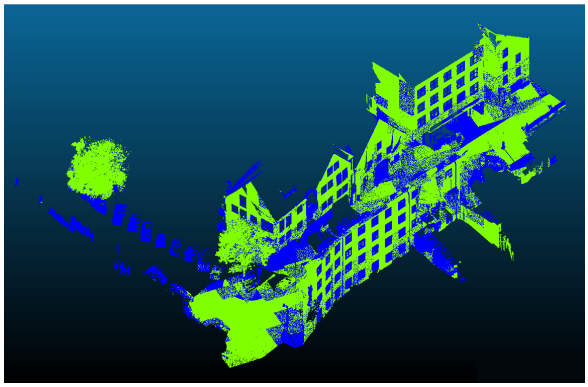
We also observe that the class `Door` is frequently labelled as `WallSurface` leading to a 0% precision and 100% false-negative-rate. This might originate in the often indistinguishable boundaries of the class to the surrounding wall structure. This might also be the reason for many `Window`s and `BuildingInstallation`s, which were segmented as part of the wall. After all, even with the aid of Focal Loss, the imbalance (see Figure 14 in the Appendix) still seems to impact the network enough to predict those objects as `WallSurface`.
Another problem could be the number of points in the sample, which we set to 12k randomly selected points in a cubed subsection of the original cloud. This exceeds the amount used in the PointNet++ paper, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space" by Charles R. Qi et al. [23], which uses only 1,024 points per pass for training and testing.
Although this may be a cause of loss in precision, we found having too few points or too small-sized training samples to be more problematic. But this would have to be inspected and analysed further.



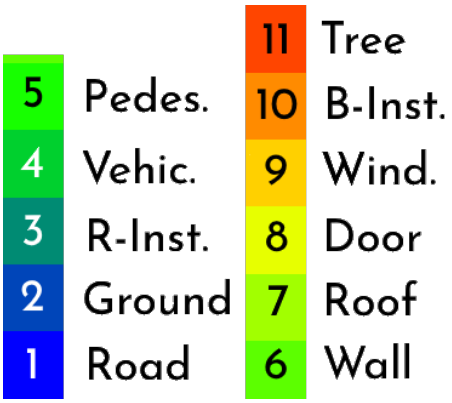
(a) True Labels



(b) Predicted Labels



(c) Deviation of the Predictions (blue)



(d) Class Legend for (a) / (b)

Figure 6: Prediction & deviation of the PointNet++ segmentation

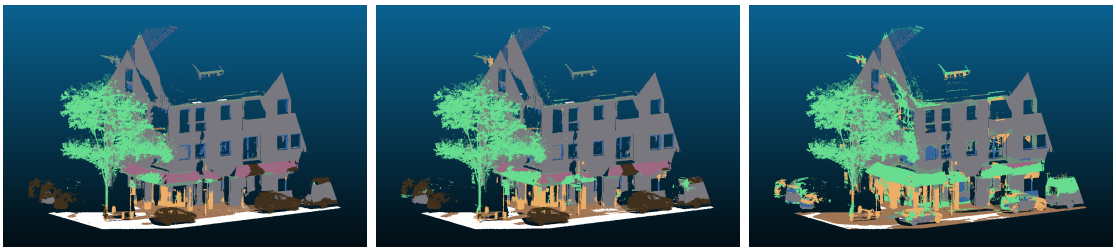
Given the suboptimal performance of PointNet++ in our case, we decide to focus our analysis on KPConv, which produces more promising results.

5.3.2 KPConv

We evaluate the results on our validation and test split for real-world data with IoU. The models perform especially well on flat surfaces such as `RoadSurface`, `GroundSurface`, and `WallSurface`. However, it is not the case for `Door`. We observe that doors are often segmented as `WallSurface`. It might be caused by the class imbalances between `WallSurface` and `Door` (see Appendix 7.8) and the similar structures between them. Since they all have similar vertical flat plane structures, the models can easily get confused and recognize them as same class. With much more `WallSurface` in the data, the models then have higher probabilities to predict these flat surfaces as `WallSurface`.
Same situation above also happens in the classes `RoadSurface` and `GroundSurface`: Due to different proportions for ground and road, synthetic data has considerably more ground points than the real-world data. When the ratio of synthetic data increases, it starts to bias the performances.
The performance on class `Window` decrease as synthetic data increase. In real-world data, windows usually contain non-flat structures such as window frames. In synthetic point

cloud, windows are usually without frames and fitted with the walls, which simplifies them to flat surfaces and makes it hard for the models distinguish between windows and walls.
The poor performances of `Pedestrian` most likely result from the lack of training data. On the other hand, `Tree` segmentation performs well, which might result from its unique structure that no other class possesses.
The objects in `BuildingInstallation` class often consist of different parts that have similar structure to other classes. The current synthetic data `BuildingInstallation` is still using only basic shapes such as cylinders or boxes, hence not close to real-life data. For example, some of them contain vertical flat surfaces that are often segmented as walls, and some of them are formed in irregular shapes that resemble noise.
Overall, the synthetic point clouds are smooth and contain relatively less significant structures for each class, which makes it hard for the model to extract meaningful features. However, we observe that the models actually still produce acceptable results with a mixture of 75%-25% between real-world data and synthetic data. We provide several visualizations for the behaviours discussed above in Appendix 7.8.
Besides IoU, we present the results with visualizations. Although there are some perturbations and noise in some objects, the models actually provide quite reasonable segmentation visually. We provide a larger view of segmentation result on validation set with training on 75% real-world data and 25% synthetic data in Appendix 7.8. Many objects are correctly recognized. However, the precise borders of these objects do not match perfectly with the true labels. Thus, highly affect the performance of IoUs.
Similar to the results from Paris-Carla-3D[6], we observe poor performance when doing transfer learning from synthetic data to real data. We attach the validation and test results in Appendix7.8.

(a) True Label (b) 100%-0% (c) 75%-25%



(d) 50%-50% (e) 25%-75% (f) 0%-100%

Figure 7: Visualization of segmentation results on validation set with different training data ratios.

Table 5: Class-wise IoUs and mean IoU on real test data with training ratio: real%-synthetics%

	100%-0%	**75%-25%**	**50%-50%**	**25%-75%**	**0%-100%**
`RoadSurface`	0,93	0,94	0,90	0,92	0,55
`GroundSurface`	0,73	0,69	0,62	0,66	0,35
`RoadInstallation`	0,51	0,35	0,40	0,23	0,23
`Vehicle`	0,67	0,66	0,51	0,48	0,00
`Pedestrian`	0,00	0,00	0,00	0,00	0,00
`WallSurface`	0,71	0,69	0,73	0,69	0,62
`RoofSurface`	0,01	0,44	0,65	0,20	0,66
`Door`	0,04	0,00	0,00	0,00	0,00
`Window`	0,24	0,24	0,30	0,17	0,21
`BuildingInstallation`	0,80	0,11	0,10	0,08	0,13
`Tree`	0,89	0,45	0,88	0,85	0,44
mIoU	0,45	0,45	0,46	0,39	0,29

6 Conclusion and Outlook

In this project, we succeed in accomplishing the four contributions of (1) defining a set of semantic classes to describe the urban environment, (2) generating synthetic point clouds including semantic information, (3) deriving of a meaningful comparison metric to express similarity between the generated synthetic data and corresponding real-world data, and (4) investigating the domain gap between real-world and synthetic data by training semantic segmentation networks on different mixtures of real-world and synthetic data.
By observing occurrences of the highly detailed class types in CityGML and OpenDRIVE and merging them into broader categories, we define a class list both suitable for the description of the urban environment as well as for semantic road space segmentation. As we keep relevant features such as windows, doors and roofs, the classes list is detailed enough for a sophisticated description of the urban environment. By merging other CityGML and OpenDRIVE class types that are less relevant to our application, we ensure the number of classes is adequate to be applicable for training semantic road-space segmentation networks.
Our synthetic data pipeline successfully generates a synthetic model of the city center of Ingolstadt based on the road network and LOD3 data provided by 3D Mapping Solutions within the SAVeNoW project [29, 32]. Using CARLA [7], we have created a synthetic Ingolstadt point cloud dataset.
The derived distances and metrics can indeed help in understanding which synthetic point clouds are better fitting the real-world data. By performing change detection in a class-wise manner using class-specific weights and combining it with the Cloud-to-Cloud distance and the mIoU value, we combine established methods already known in the field in a novel way to construct our metric. Furthermore, our distance measure gives a physically meaningful result helping with quantifying geometric divergence between two point clouds.
With the experiments on training semantic segmentation network with different mixtures of real-world to synthetic data, we gain precious insights in the feasibility as well as difficulties in fusing insufficient training data set with synthetic data. We observe that the overly-smooth and featureless structure of synthetic data is the main cause of the gap between real and synthetic data domains. Moreover, we find that although the synthetic data, due to its simplified structure, cannot entirely replace real-world training data, it still serves as an effective option for training a segmentation network with acceptable results given the ratio of synthetic-to-real-world data is not too high.

Outlook and Future Research Directions Currently, the synthetic map still requires 3 DoF alignments. The synthetic model could fit better with real-world data by applying further alignment methods, leading to a better synthetic point cloud. Furthermore, the meshes production from our method is relatively simple. More complex methods could be applied here to generate better details and realistic meshes. Otherwise, since the models and the real-world data are aligned globally, a trained mesh deformation for the models to fit the point cloud better could also be used.
For the synthetic data collection, currently, we inject a small amount of noise into the point cloud. For the future, we propose to research how to make this noise more realistic; for instance, by using a drop-off rate based on intensity, a drop-off rate based on other

factors or even a learned noise.
To ensure an equal influence between the distance and the mIoU on the metric output, we intend to further fine-tune the metric weights. So far, we have set the weight parameters empirically. In the future, a more rigorous and systematic approach can lead to a more balanced metric. Additionally, we propose to make the voxel size l_{voxel} for the IoU calculation a function of the distance measure. This is because with the mIoU, we want to penalize proper semantic mismatches, not errors that arise due to a translation shift - this shift is already taken into account with the distances. Thus, we advocate for investigations whether larger, adaptive voxel sizes (in the order of magnitude of the distance measure) can help decouple the mIoU from the geometric error. An alternative solution to the decoupling problem could be to perform cloud registration such as Iterative Closest Point as a preprocessing step for the mIoU calculation (note that registration should be performed only after having obtained the distances).
With the current experiment settings, we see the possibility to successfully train segmentation networks with synthetic data. However, the approaches we use and results we have obtained so far still have considerable room for improvements. Our approach to mix the real-world and synthetic data is relatively naive. With random down-sampling approach, the 3D point clouds' geographic structures are not well preserved. Moreover, this method does not guarantee that the selected points lie in different areas of the two point clouds. The two down-sampled point clouds thus may contain highly overlapping areas when merging. An alternative way would be spatially cutting and merging point clouds in CloudCompare to ensure selection of different areas for each point cloud. Further, we do not focus on model-tuning in our work, since our goal is to assess the trainability of data. It may be worthwhile to pursue hyperparameter tuning in order to fully uncover the potential of synthetic point clouds. Last but not least, the question of whether it is possible to bridge the domain gap between real and synthetic data, and if possible, how should the problem be approached, is yet to be answered.
Finally, we are curious to see how our approach can be extended by incorporating other modalities such as camera and radar data, effectively paving the way for synthetic multimodal perception and data generation.

Bibliography

[1] Iro Armeni, Sasha Sax, Amir. R. Zamir, and Silvio Savarese. "Joint 2D-3D-Semantic Data for Indoor Scene Understanding". In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.01105 [cs.CV].

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences". In: *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11618–11628. DOI: 10.1109/CVPR42600.2020.01164.

[4] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[5] *CloudCompare*. https://www.cloudcompare.org/. (Visited on 07/20/2023).

[6] Jean-Emmanuel Deschaud, David Duque, Jean Pierre Richa, Santiago Velasco-Forero, Beatriz Marcotegui, and François Goulette. "Paris-CARLA-3D: A Real and Synthetic Outdoor Point Cloud Dataset for Challenging Tasks in 3D Mapping". In: *Remote Sensing* 13.22 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13224713.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 1–16. URL: https://proceedings.mlr.press/v78/dosovitskiy17a.html.

[8] Amirata Ghorbani, Vivek Natarajan, David Coz, and Yuan Liu. "DermGAN: Synthetic Generation of Clinical Skin Images with Pathology". In: *Proceedings of the Machine Learning for Health NeurIPS Workshop*. Ed. by Adrian V. Dalca, Matthew B.A. McDermott, Emily Alsentzer, Samuel G. Finlayson, Michael Oberst, Fabian Falck, and Brett Beaulieu-Jones. Vol. 116. Proceedings of Machine Learning Research. PMLR, 13 Dec 2020, pp. 155–170.

[9] Daniel Girardeau-Montaut, Michel Roux, Raphaël Marc, and Guillaume Thibault. "Change detection on point cloud data acquired with a ground laser scanner". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36 (Jan. 2005).

[10] Gerhard Gröger, Thomas H. Kolbe, Claus Nagel Nagel, and Karl-Heinz Häfele. *OGC City Geography Markup Language (CityGML) En-coding Standard*. https://portal.ogc.org/files/?artifact_id=47842. [Online; accessed Jun 2023]. 2012.

[11] Andreas Haigermoser, Bernd Luber, Jochen Rauh, and Gunnar Gräfe. "Road and track irregularities: Measurement, assessment and simulation". In: *Vehicle System Dynamics* 53 (May 2015), pp. 1–80. DOI: 10.1080/00423114.2015.1037312.

[12] Abderrazzaq Kharroubi, Florent Poux, Zouhair Ballouch, Rafika Hajji, and Roland Billen. "Three Dimensional Change Detection Using Point Clouds: A Review". In: *Geomatics* 2 (Oct. 2022), pp. 457–486. DOI: 10.3390/geomatics2040025.

[13] Moritz Klischat and Matthias Althoff. "Generating Critical Test Scenarios for Automated Vehicles with Evolutionary Algorithms". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 2352–2358. DOI: 10.1109/IVS.2019.8814230.

[14] Dimitri Lague, Nicolas Brodu, and Jérôme Leroux. "Accurate 3D comparison of complex topography with terrestrial laser scanner: Application to the Rangitikei canyon (N-Z)". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 82 (2013), pp. 10–26. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2013.04.009.

[15] *laspy*. https://github.com/laspy/laspy. [Online; accessed Jul 2023].

[16] Patrick Madlindl. *CloudMetrics Repository*. https://github.com/Madlab2/CloudMetrics. [Online; accessed Jul 2023]. July 2023.

[17] Mathworks. https://de.mathworks.com/products/roadrunner.html. [Online; accessed Jul 2023]. 2018.

[18] Duc Nguyen, Florian Hauck, Yan-Ling Lai, Patrick Madlindl, and Xinyuan Zhu. July 2023. URL: https://gitlab.lrz.de/di-lab-st-2023/documentation.

[19] NVIDIA. *NVIDIA Omniverse Replicator*. https://developer.nvidia.com/omniverse/replicator. [Online; accessed Jul 2023]. 2022.

[20] ASAM OPENDRIVE. https://www.asam.net/standards/detail/opendrive/. [Online; accessed Jul 2023]. 2021.

[21] py4dgeo Development Core Team. *py4dgeo: library for change analysis in 4D point clouds*. https://github.com/3dgeo-heidelberg/py4dgeo. [Online; accessed Jul 2023].

[22] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[23] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 5105–5114. ISBN: 9781510860964.

[24] Jean Pierre Richa, Jean-Emmanuel Deschaud, François Goulette, and Nicolas Dalmasso. "AdaSplats: Adaptive Splatting of Point Clouds for Accurate 3D Modeling and Real-Time High-Fidelity LiDAR Simulation". In: *Remote Sensing* 14.24 (2022). ISSN: 2072-4292. DOI: 10.3390/rs14246262.

[25] Canada Safe Software of British Columbia. https://community.safe.com/s/question/0D54Q000080hHBiSAM/performance-benchmarking. [Online; accessed Jul 2023]. 1996.

[26] Benedikt Schwab, Christof Beil, and Thomas H. Kolbe. "Spatio-Semantic Road Space Modeling for Vehicle–Pedestrian Simulation to Test Automated Driving Systems". In: *Sustainability* 12.9 (2020). ISSN: 2071-1050. DOI: 10.3390/su12093799.

[27] 3D Mapping Solutions. https://www.3d-mapping.de/home/. [Online; accessed Jul 2023].

[28] Steven Spiegel and Jorge Chen. "Using Simulation Data From Gaming Eenvironments For Training a Deep Learning Algorithm on 3D Point Clouds". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* VIII-4/W2-2021 (2021), pp. 67–74. DOI: 10.5194/isprs-annals-VIII-4-W2-2021-67-2021.

[29] Gerhard Stanzl. *SAVeNoW*. https://savenow.de/de/. [Online; accessed Jul 2023].

[30] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. "KPConv: Flexible and Deformable Convolution for Point Clouds". In: *Proceedings of the IEEE International Conference on Computer Vision* (2019).

[31] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[32] Olaf Wysocki, Sophie Haas Goschenhofer, and Benedikt Schwab. https://github.com/savenow/lod3-road-space-models. [Online; accessed Jul 2023]. 2021.

[33] Olaf Wysocki, Ludwig Hoegner, and Uwe Stilla. "TUM-FAÇADE: Reviewing and Enriching Point Cloud Benchmarks for Façade Segmentation". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLVI-2/W1-2022 (2022), pp. 529–536. DOI: 10.5194/isprs-archives-XLVI-2-W1-2022-529-2022. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLVI-2-W1-2022/529/2022/.

[34] Aoran Xiao, Jiaxing Huang, Dayan Guan, Fangneng Zhan, and Shijian Lu. "Transfer Learning from Synthetic to Real LiDAR Point Cloud for Semantic Segmentation". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.3 (June 2022), pp. 2795–2803. DOI: 10.1609/aaai.v36i3.20183.

[35] Vivien Zahs, Lukas Winiwarter, Katharina Anders, Jack G. Williams, Martin Rutzinger, and Bernhard Höfle. "Correspondence-driven plane-based M3C2 for lower uncertainty in 3D topographic change quantification". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 183 (2022), pp. 541–559. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2021.11.018.

[36] Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. "An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation". In: *Remote. Sens.* 8 (2016), p. 501.

[37] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[38] Örkény Zováthi, Balázs Nagy, and Csaba Benedek. "Point cloud registration and change detection in urban environment using an onboard Lidar sensor and MLS reference data". In: *International Journal of Applied Earth Observation and Geoinformation* 110 (2022), p. 102767. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2022.102767.

List of Authors

Patrick Madlindl: Abstract, 1, 3.0, 3.2, 3.3, 4.0, 4.1, 5.0, 5.2, (topic-related) 6, 7.2, 7.2.1, 7.3, 7.6
Duc Nguyen: (minor) Abstract, 3.4, 3.5, 6 (Outlook regarding Map generation and noise injection)
Xinyuan Zhu: 4.2.1(PointNet++ part), 4.2.2(first paragraph), 5.3.1, 7.1(Class list creation and definition)
Florian Hauck: 4.2.1, 4.2.2(Data Preparation & Training), 5.3.1 (Text, Figures), 7.7
Yan-Ling Lai: 3.4.1(r:trån), 3.4.2, 3.4.3, 4.2, 4.2.1(Intro and KPConv), 4.2.3, 5.3.2, 6(Segmentation network training)

Individual Contributions of Project Members

All project members have had an equal share in common tasks such as documentation, labelling the MLS data, creating presentation slides and writing this report. In the following, we list the particular contributions that each member has achieved on top of that.

Nguyen Duc Researching and engineering software for synthetic data pipelines, Taking part in producing meshes and setup semantics for map generation, Editing map from mesh components and setup road logic in CARLA, researching and implementing the extension of manual control vehicle and noise injection, researching about domain gap between synthetic and real data in order to refine and align synthetic point cloud generation with support from Patrick Madlindl, supporting implement adaptation of KPConv and PointNet++ for new datasets, Researching loss for imbalanced point cloud, supporting training and analyzing results of KPConv, Coordinating problem between tasks and providing support wherever needed.

Florian Hauck has been researching & engineering software for synthetic data pipelines, taking part in producing meshes and set up semantics for map generation, editing the map from mesh components and setting up road logic in CARLA with Nguyen Duc. Utilizing RoadRunner and UnrealEngine Editor to help setting up the environment used for the synthetic point cloud generation. Main role in setting up a custom PointNet++ model, data preprocessing and dataset splitting. Researching possibilities for imbalanced datasets and investigating KPConv implementation with Yan-Ling. Generating and refining results for the PointNet++ model and analyzing its results. Not to forget the manual labelling of the real-world test data split.

Yan-Ling Lai Working on the early stage data survey, and investigate the properties of feature types in CityGML LOD3 building and OpenDRIVE data. Participating in the design and formulation of proper class list for labelling and training. Assisting in the creation of mesh files for custom map in Carla by leveraging FME and r:trån to transform CityGML and OpenDRIVE data. Investigating KPConv implementation and integrate the repository with data set. Developing data transform and mixing procedure,

and conduct model training. Organizing training results of KPConv. Analyzing training results and providing insights and explanations.

Patrick Madlindl has been the responsible for exploring different labelling tools and methods and designed the labelling process. Also, Patrick Madlindl has been highly involved in creating the class list and definitions. Further, he has been responsible for data set statistics and the corresponding Python implementation. Patrick Madlindl has researched the different cloud comparison methods, distances and metrics and has had the responsibility to design the metric and deterministically compare synthetic with real-world point clouds. He implemented the corresponding Python script.

Xinyuan Zhu Survey the properties and features from the provided OpenDRIVE and CityGML dataset and creating the class list and definitions based on the investigation. Helps to explore the function of FME software. Research and implement the PointNet++, transform and preprocessing the dataset into the target form, helps to train the network, fix the issues while transferring the dataloader from KPConv, set up the training container on the server. And analyze the performance of the PointNet++.

7 Appendix

Here, the interested reader can find additional information and further insightful details on the project that could not be included in the main part of the report due to page limits.

7.1 Class List and Definitions

In order to start labelling the real-world point clouds and train the semantic segmentation network, we need to create a list containing all the classes, and formal definition of each class, so that we could minimize the inconsistency when we separately labelling. Meanwhile, we want to keep the numbers of the objects in each class to be roughly identical. So our first step is to view the dataset we have, CityGML and OpenDRIVE, in the FME Workbench. As described in the main part of the paper, the FME Workbench could provide us various information about the properties of the dataset, e.g. the names and the numbers of different types within the dataset. So, based on these information, we listed all the types we had so far in CityGML and OpenDRIVE dataset as in Figure 9, then we try to merge the similar or same types and try to keep the numbers of objects in different types roughly same, so that we could avoid data imbalance in the final dataset. Since our project are focusing on the building related objects, so although for some classes like Road Installations and Trees, they have very few objects comparing to other classes, we could ignore this point in our project. After considering the points mentioned above, we define a class list as shown in Figure 8.

Description	Label No.	Labelling Class
Category 1: 2D Object on the Ground		
Driveable area (road, parking spaces), traffic islands, street markings of all kinds (traffic signs, lanes, ...)	1	Road Surface
Human walkable horizontal surface. Includes pavement, curb stones, stairs leading to doors, "conventional stairs", vegetations that are not trees (bushes, flowers, grass...), but not vegetation close to the stem of a tree. Excludes drivable area.	2	GroundSurface
Category 2: 3D Road Object - Static		
All kinds of static objects in the road area: poles, traffic lights, street lamps, benches, controllers boxes, fences (on the ground), advertising signs, "Litfaßsäule"... Does not include objects that stand directly in front of a house such as flower pots.	3	Road Installations
Category 3: 3D Road Object - Dynamic		
Traffic participants, whether moving or standing still: Cars, bicycles, motorbikes, lorries, buses... Includes drivers/passengers. Excludes pedestrains.	4	Vehicle
Human beings, pedestrians. All humn beings except cyclists, motor-bikers, car/lorry/bus drivers/passengers.	5	Pedestrian
Category 4: City Environment		
Perpendicular surfaces in street area/building area. Excludes windows, doors.	6	WallSurface
Roofs	7	RoofSurface
Doors. Includes door frames/portals, and some small part of the wall surrounding the door.	8	Door
Windows. Includes window frames, some small part of the wall surrounding the window. Does not include objects in the window such as curtains, flower pots.	9	Window
Ceiling (Non-human-walkable horizontal surface), window awning (The small window roof), windowsill, balcony (Includes Poles, fencing on the balcony), decor, building add-ons (Some add-ons and chimneys on the roofs and building walls), rain pipes, dormer (excluding dormer window), Objects very close to buildings such as flower pots. Does not include objects in the building that are observed through windows.	10	Building Installation
Category 5: Nature & Noise		
Trees. Includes bushes/vegetation close to the stem of a tree.	11	Tree
Any other points that cannot be classified, e.g. multipath window reflections, noisy points, points inside a building (e.g. interior walls, layers of window reflections, all objects inside buildings/windows, e.g. flower pots, curtains)	12	Noise

Figure 8: Class list with definitions used for labelling the MLS data and training segmentation networks.

Labelling Class	LOD3	OpenDrive Converted (LOD2)	Numbers (LOD3+LOD2)
	X	Lateral Filler Surface/Longitudinal Filler Surface Within Road (AuxiliaryTrafficArea)	(2073+366)
Road Surface	X	No Parking Area(AuxiliaryTrafficArea)	4
GroundSurface	GroundSurface+OuterFloorSurface	GroundSurface	(65+133)+228
Road Installations	X	TrafficLight(CityFurniture) , TrafficSign+Gefahrzeichen(CityFurniture) , Sinnbild(CityFurniture) , Richtzeichen(CityFurniture) , Verkehrseinrichtungen(CityFurniture) , Vorschriftzeichen(CityFurniture) , Zusatzzeichen(CityFurniture) , StreetLamp(CityFurniture) , Pole(CityFurniture) , Bench(CityFurniture) , ControllerBox(CityFurniture)	98
Vehicle	X		
Pedestrian			
	WallSurface+ClosureSurface	WallSurface	(2425+12)+1158
WallSurface	X	Wall(CityFurniture)	27
RoofSurface	RoofSurface	RoofSurface	422+228
Door	Door	X	61
Window	Window	X	1075
Building Installation	BuildingInstallation	X	
Tree	X	Tree(SolitaryVegetationObject)	262
Noise			

Figure 9: Original names of each types from CityGML and OpenDRIVE and the numbers of objects included.

7.2 Data Splits

The synthetic data generation was possible for a subset area of Ingolstadt (the area where we have been provided with LOD3 data). We split this area into a training, validation and test area. The corresponding real-world MLS point clouds were labelled and constitute the data sets training real-world, validation real-world and test real-world. The split has been chosen with three constraints in mind:

1. There should be no overlap between each of the sets to avoid training leak.
2. The balance between the sets should roughly follow a $70 - 20 - 10$ ratio between train, validate, and test.
3. Each set should contain a sufficient number of points and instances for each class that we train on.

The labelled real-world data sets are shown in Figure 12, while synthetic datasets from the same area are depicted in Figure 13.

7.2.1 Data Split Statistics

To get an objective insight into how the data ratios are, we implemented a Python script [16] that computes some data set statistics based on point counts. Point counts are no reliable indication on the number of instances (as point densities in the point clouds can vary from object to object), but can give a rough estimate. Figure 14 shows the results of these statistics. Each bar represents one class. The total height of the bar represents the share of this class over all data sets, while the colors of each bar indicate the distribution of the class points between train, validation and test in the real-world MLS data.

7.3 Parameters for M3C2

The parameters for M3C2 are found by using CloudCompare's implementation of M3C2. There, before performing M3C2, the parameters can be estimated by the programm. We adopt these parameters, but also make use of p4dgeo's option to run M3C2 with multiple `Normal_Radius` values. The parameters used in this project can be found in Table 6.

Table 6: Parameters for M3C2 in [m].

`Cyl_Radius`	`Normal_Radius`	`Max_Distance`
0.21	0.21	10
0.21	0.42	10
0.21	0.84	10

7.4 Map Generation

In the context of CARLA, map generation involves aligning meshes to create a realistic environment for synthetic data. The alignment is achieved through 3 DoF, maintaining the original rotation, and matching coordinates between paired points. To incorporate custom semantics, specific semantic tags were created for real-world data, following CARLA's guideline documentation, and updating the source code accordingly. Each mesh was then associated with a semantic tag by organizing the assets in Unreal Engine accordingly. The alignment process is further refined by manually adjusting the meshes using an anchor point or line. This step ensures precise alignment and accurate positioning of each mesh within the environment. By selecting appropriate anchor points or lines that match between meshes, the relative positions of various objects are preserved, resulting in a coherent and realistic synthetic environment. During this manual alignment process, the user carefully adjusts the meshes to fit seamlessly together, taking into account their spatial relationships and interactions with the surrounding elements. By iteratively aligning and adjusting the meshes, the synthetic environment gradually takes shape, closely resembling a real-world setting. Furthermore, we provide detail visualization for Road Installation that is processed using r:trån and FME.

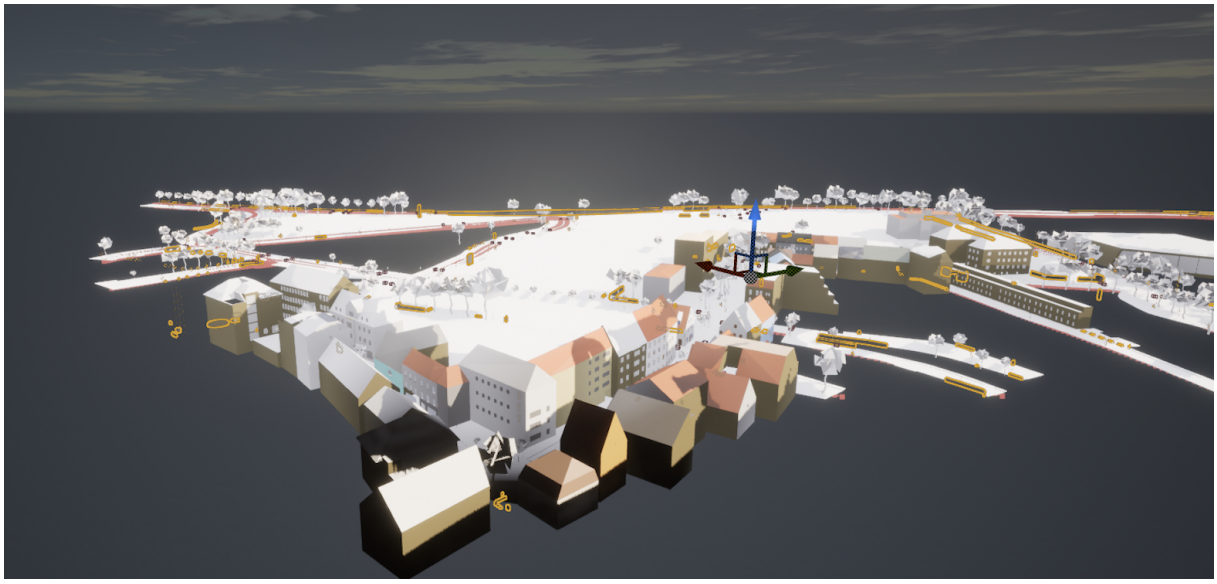


Figure 10: Synthetic custom map in Unreal-Engine 4

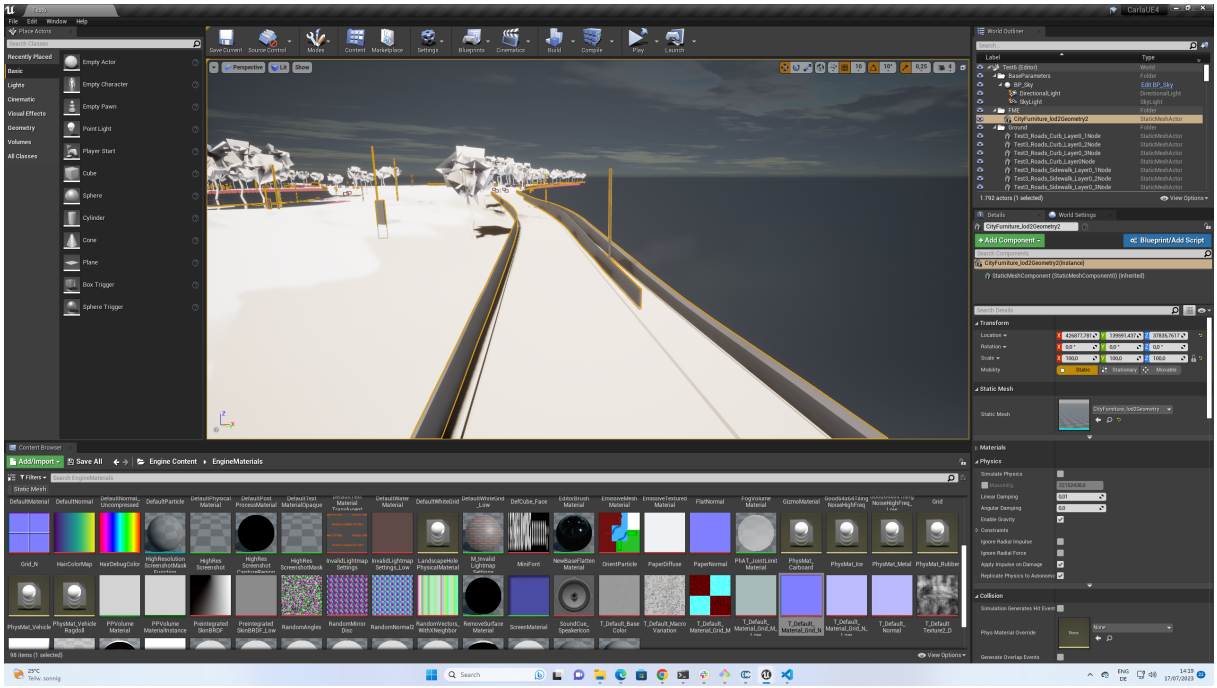


Figure 11: Road installation from FME in Unreal Engine 4

7.5 Synthetic Data Generation

In Table 7, we specify the exact LiDAR configuration used in our setup.

Table 7: LiDAR Configuration in CARLA

Attribute	**Value**
channels	128
points_per_second	500000
rotation_frequency	20
upper_fov	15
lower_fov	−25
horizontal_fov	360
range	100

7.6 Real-world and Synthetic Point Cloud Visualizations

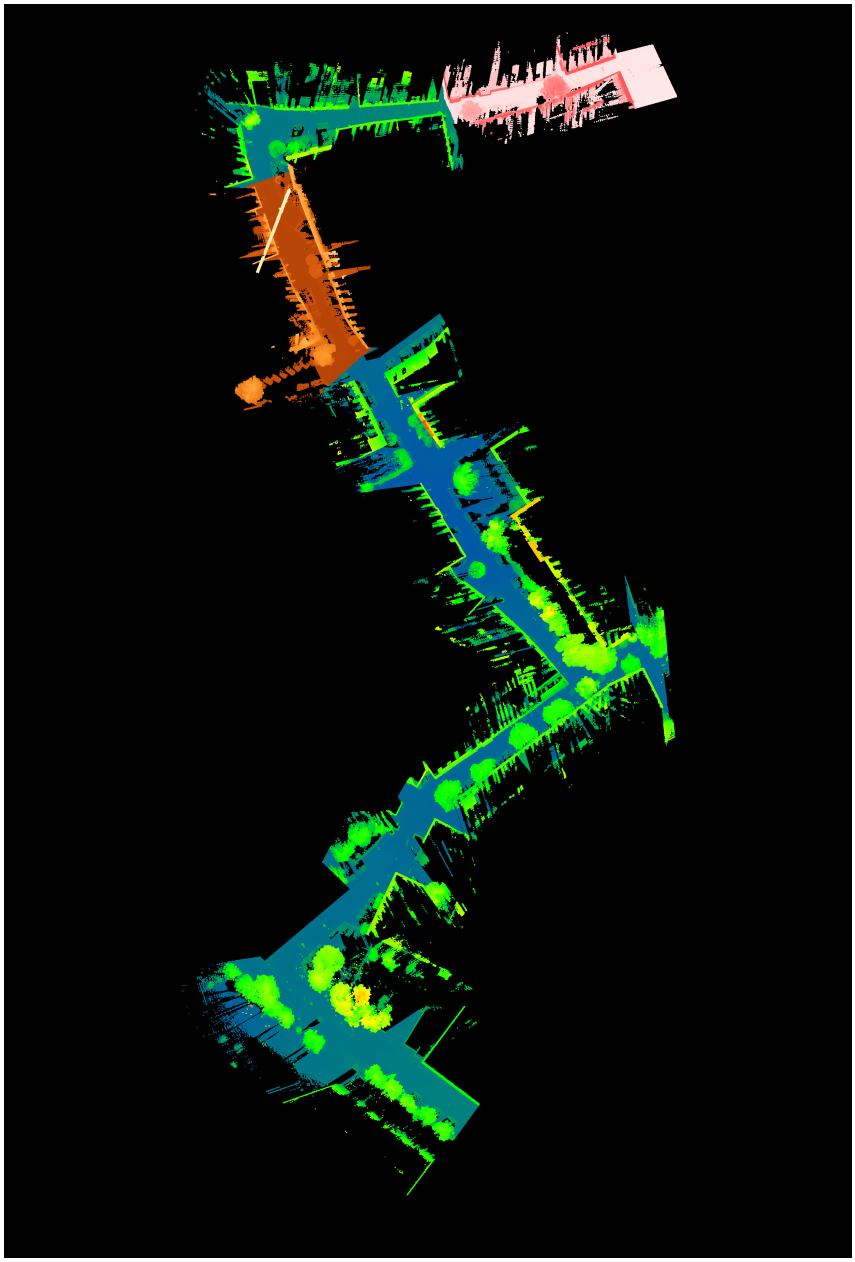


Figure 12: Data Splits of real-world MLS point clouds. Blue/green color scale: training data set. Brown/red color scale: validation data set. Pink scale: test data set.

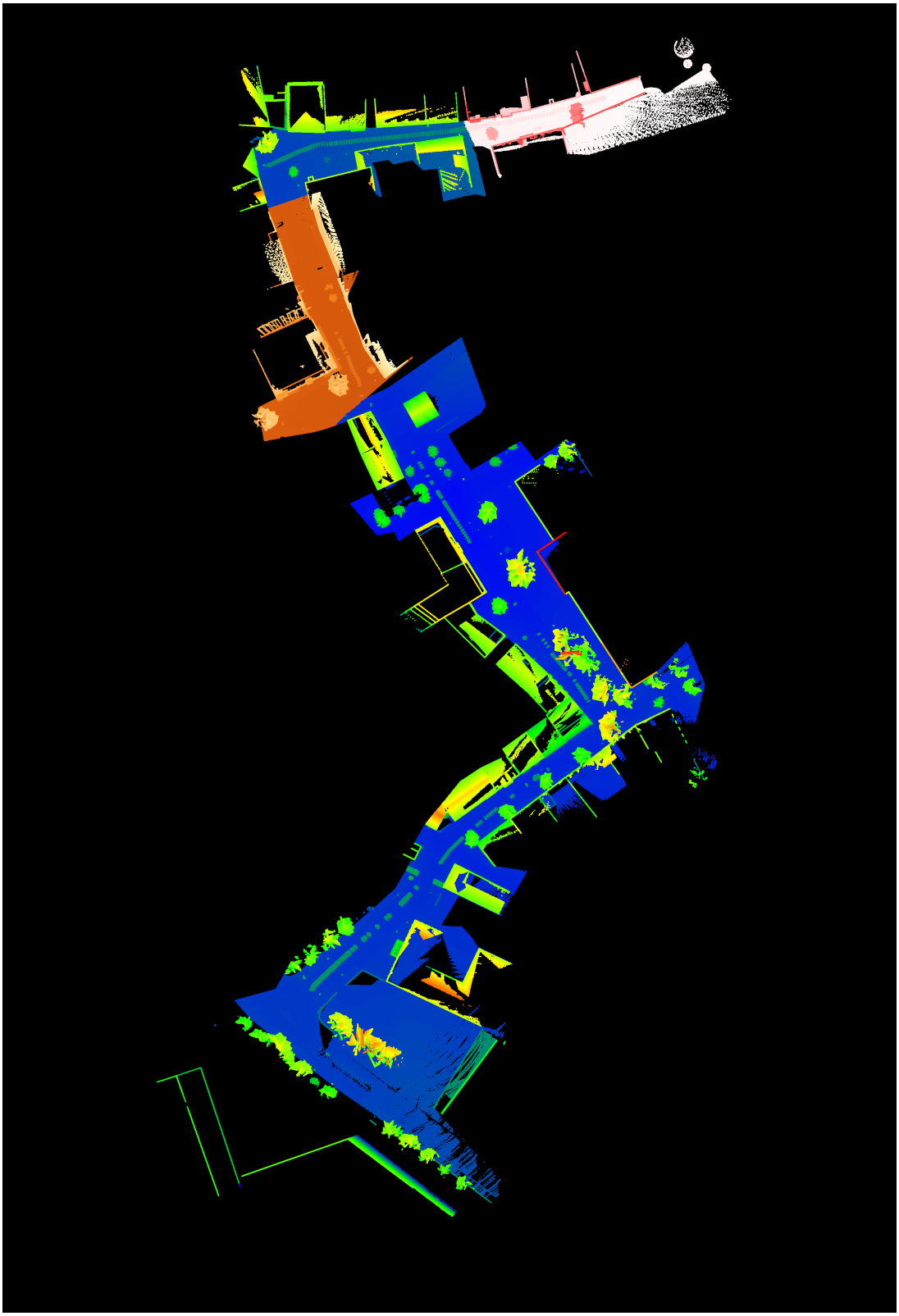


Figure 13: Data Splits of synthetic point clouds. Blue/green color scale: training data set. Brown/red color scale: validation data set. Pink scale: test data set.

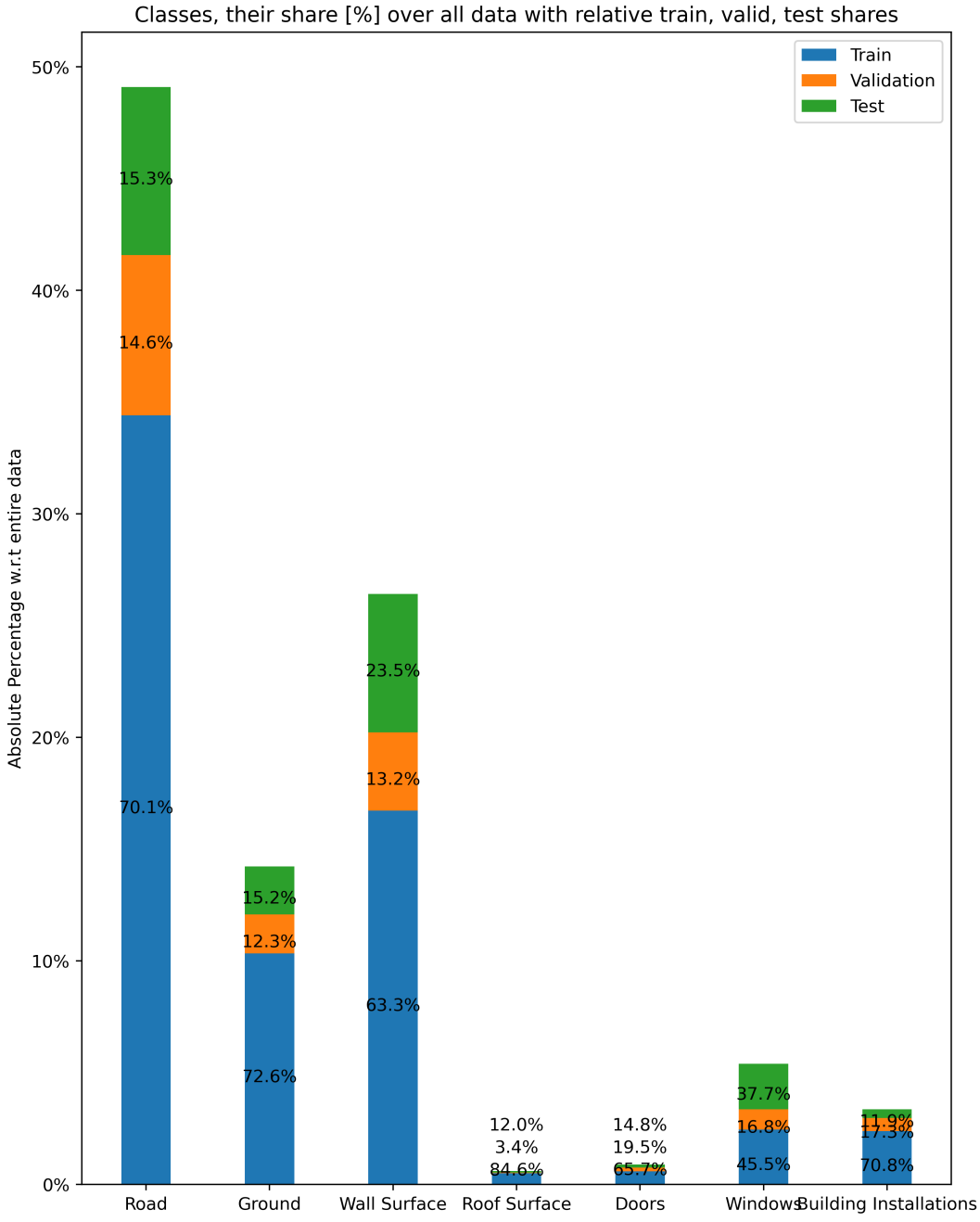


Figure 14: Statistics of data splits of real-world MLS point clouds. The height/lengths of the bars indicate the absolute percentage w.r.t the entirety of trainig, validation and test points. The percanteges written in the bars themselves indicate the relative ditribution between train, validation and test for each class. For instance, almost 50% of all points are of class road; while 70.1% of road points are wihtin the training set.

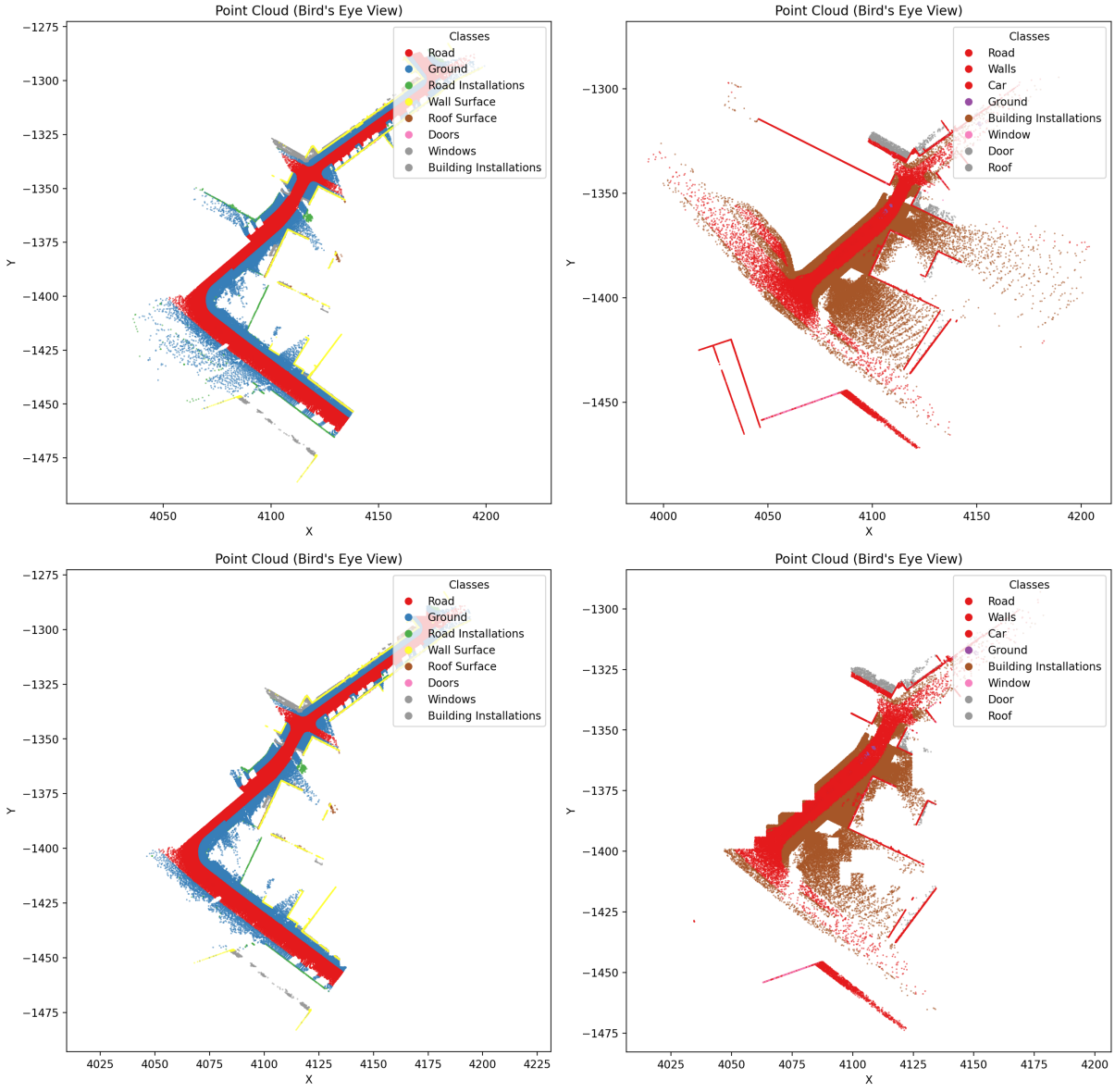


Figure 15: Grid Filter Result. Bird's eye views of real-world (left) and non-aligned synthetic (right) point clouds. Both are shifted to the same reference frame and only certain classes as indicated in the legend are displayed. Top row: unfiltered point clouds. Bottom row: grid-filtered point clouds. It can be seen that areas where there are only points from one of the two clouds are filtered out, while the main area of comparison is being kept.

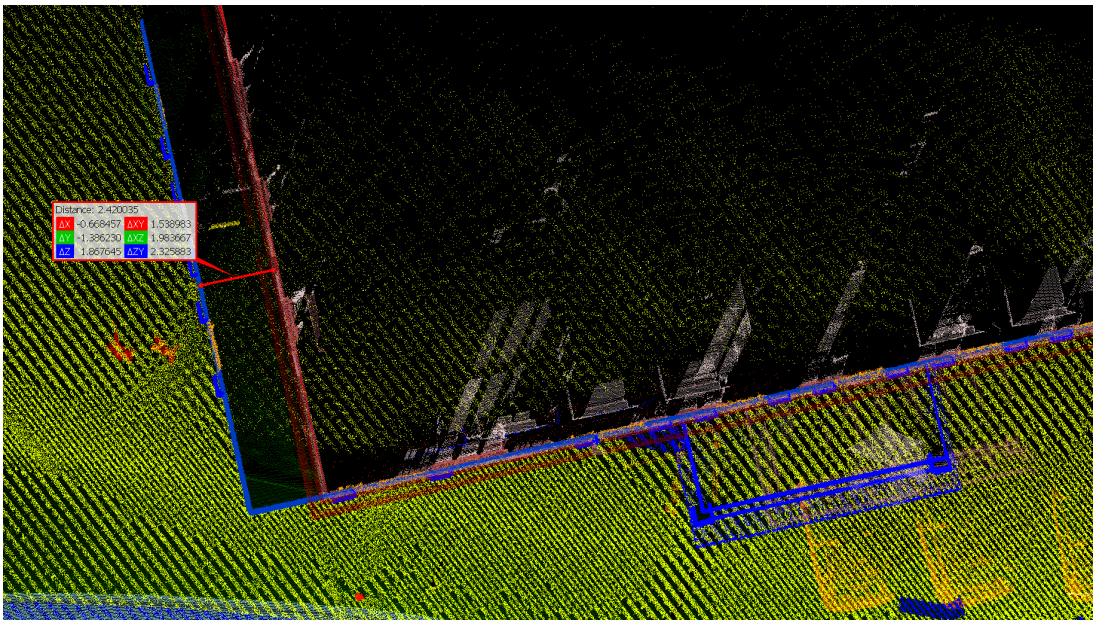


Figure 16: Top view showing a section of the unaligned synthetic and real-world point cloud displaying walls (read: real-world, blue: synthetic) of a house. The walls happen to be arranged in such a way that one wall has almost no shift in the XY plane, while the other wall is visibly shifted. Note that both walls do have notable shift in Z direction.

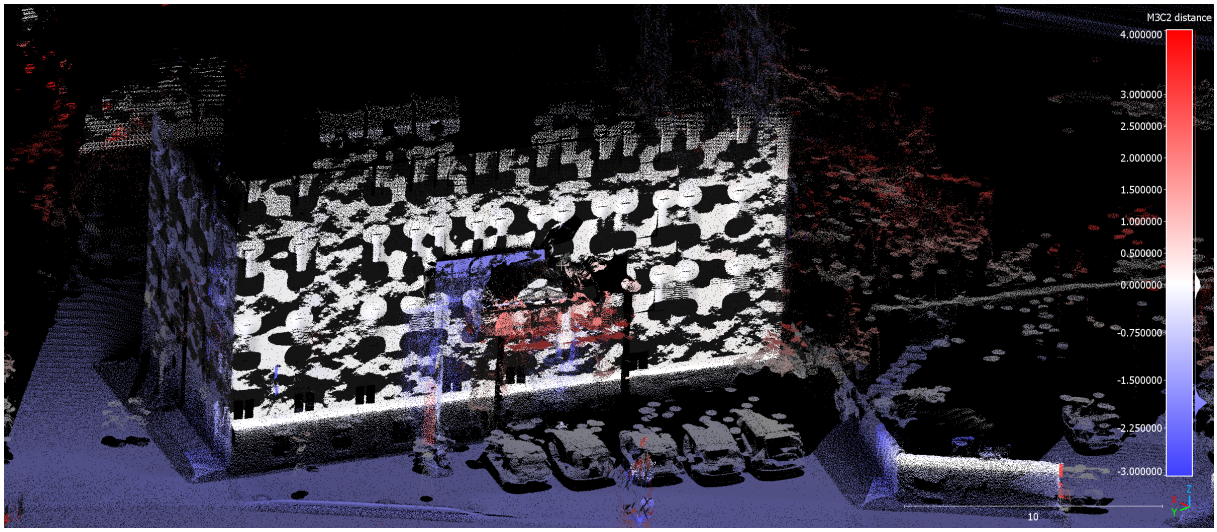


Figure 17: M3C2 distance point cloud with 3D view of the same house whose walls have been shown in Figure 16. Blue and red points have relatively high M3C2 distances, while white points have small ones. The insensitivity of M3C2 towards shifts orthogonal to the estimated normal can clearly be seen. Similar as the (deep blue) ground points, both walls are shifted by a notable amount in Z direction. However, only the side-facing wall on the left is also shifted in the XY plane (see Figure 16). The M3C2 result clearly does not consider the Z shift in the walls (which is orthogonal to the normals of the walls), resulting in small M3C2 distances for the front-facing wall, while the side-facing wall's XY shift is well reflected in the M3C2 distance.

7.7 PointNet++ Results

Table 8: Confusion Matrix PointNet++ on Validation Set

True Label \ Predicted	Road	Ground	RoadInst.	Vehicle	Pedestrian	Wall
RoadSurface	655345	139695	0	5182	0	17320
GroundSurface	94825	130538	4	73	0	61506
RoadInst.	6666	37713	412	20652	0	96240
Vehicle	14637	31391	2938	47497	0	80680
Pedestrian	10490	8284	100	611	0	9692
WallSurface	11076	56746	789	9255	64	2451781
RoofSurface	0	14	0	0	0	22251
Door	0	3555	69	506	0	37796
Window	1431	5558	2	1078	0	606316
Build.Inst.	0	9212	616	732	0	244621
Tree	200	14749	50	1368	6	119998
Precision (%)	82.46	29.84	8.27	54.62	0.00	65.41

True Label \ Predicted	Roof	Door	Window	Build.Inst.	Tree	FNR (%)
RoadSurface	0	0	4148	0	55404	25.28
GroundSurface	0	0	10846	246	60013	63.54
RoadInst.	4003	0	15664	2190	52500	99.83
Vehicle	2663	0	31950	8245	276035	90.42
Pedestrian	148	0	1790	519	9190	100.00
WallSurface	7495	0	22468	26111	261257	13.88
RoofSurface	0	0	209	337	1078	100.00
Door	174	0	302	74	17547	100.00
Window	52	0	4996	4005	72980	94.46
Build.Inst.	1515	0	26625	7492	78425	97.97
Tree	44	0	25360	7758	1081802	4.38
Precision (%)	0.00	0.00	3.46	13.15	55.02	

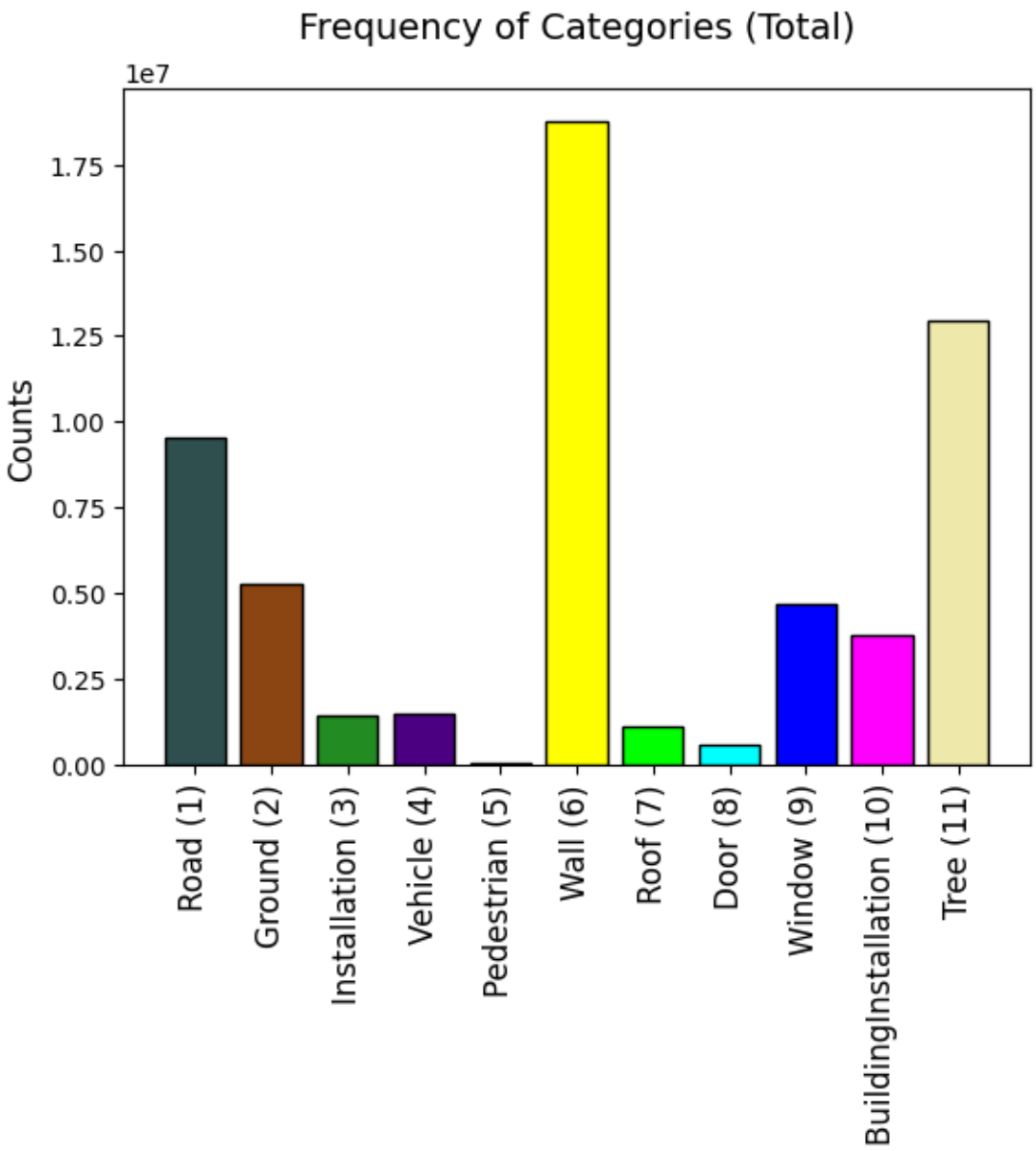


Figure 18: Shifted class distribution after preprocessing for PointNet++

7.8 KPConv Results and Visualizations

Table 9: Class-wise IoUs and mean IoU on real validation data with training ratio: real%-synthetics%

	100%-0%	**75%-25%**	**50%-50%**	**25%-75%**	**0%-100%**
RoadSurface	0,90	0,89	0,89	0.87	0,55
GroundSurface	0,58	0,50	0,49	0.42	0,31
RoadInstallation	0,34	0,27	0,20	0.31	0,00
Vehicle	0,89	0,52	0,57	0.72	0,00
Pedestrian	0,00	0,00	0,00	0,00	0,59
WallSurface	0,76	0,71	0,66	0.73	0,21
RoofSurface	0,16	0,09	0,16	0.20	0,00
Door	0,17	0,00	0,00	0,00	0,28
Window	0,48	0,45	0,28	0.28	0,08
BuildingInstallation	0,25	0,18	0,00	0.11	0,45
Tree	0,94	0,93	0,90	0.82	0,01
mIoU	0,50	0,41	0,39	0,36	0,22

Table 10: Class-wise IoUs and mean IoU on real validation data with training ratio: real%-synthetics% (For transfer learning, it is pre-trained with all synthetic data, and fine-tuned on real train data.)

	100%-100%	**Transfer learning**
RoadSurface	0,90	0,89
GroundSurface	0,58	0,50
RoadInstallation	0,34	0,27
Vehicle	0,89	0,52
Pedestrian	0,00	0,00
WallSurface	0,76	0,71
RoofSurface	0,16	0,09
Door	0,17	0,00
Window	0,48	0,45
BuildingInstallation	0,25	0,18
Tree	0,94	0,93
mIoU	0,50	0,41

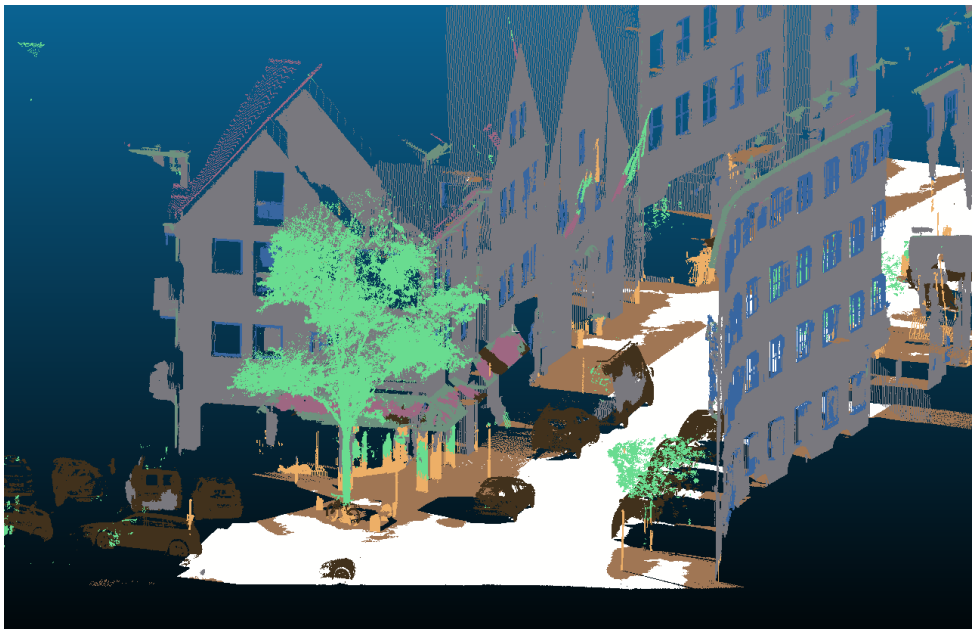
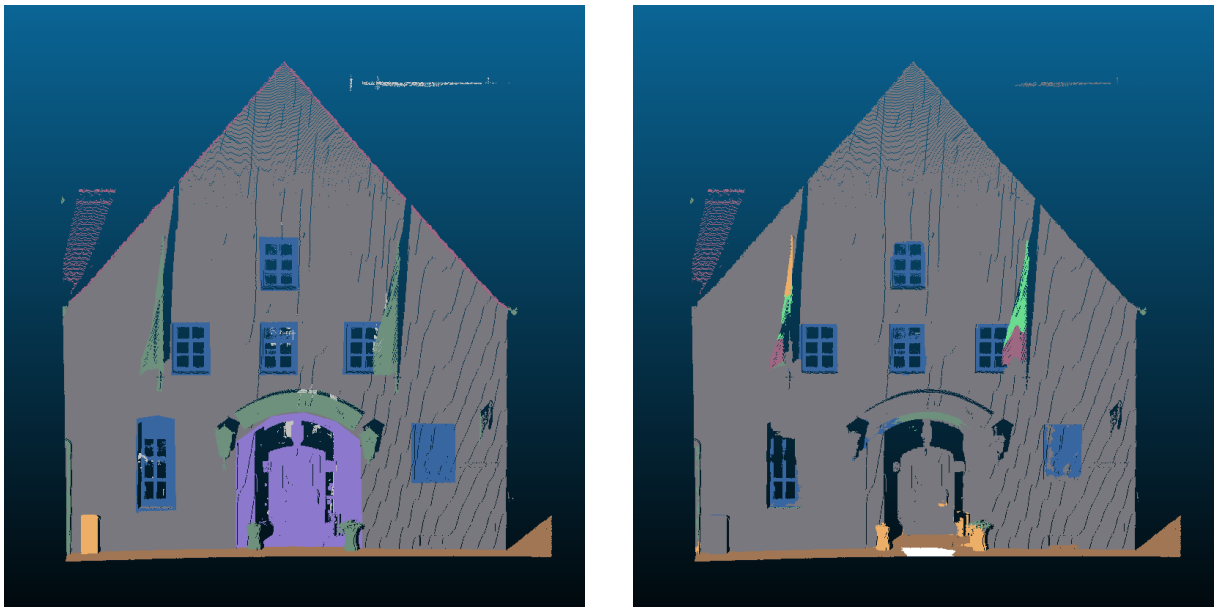


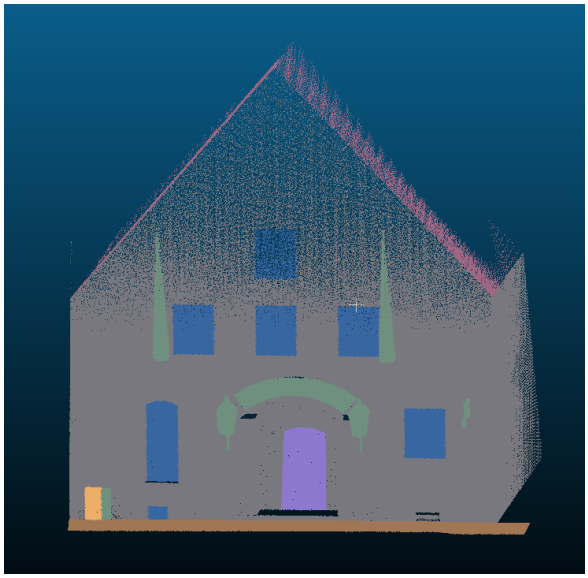
Figure 19: Segmentation result on validation set with training on 75% real and 25% synthetics



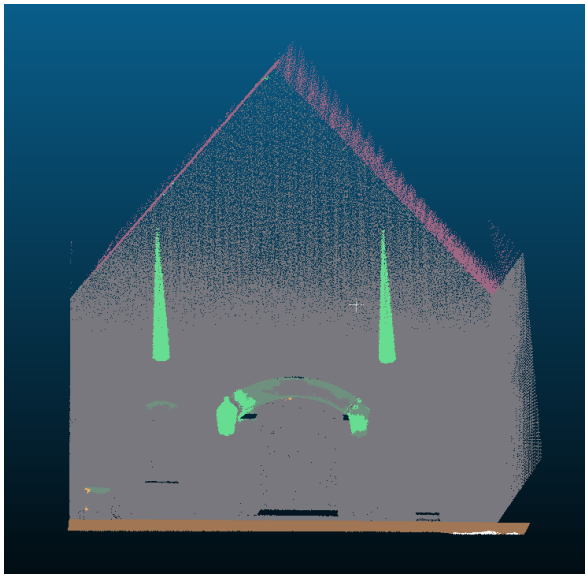
(a) True class label

(b) Predicted labels

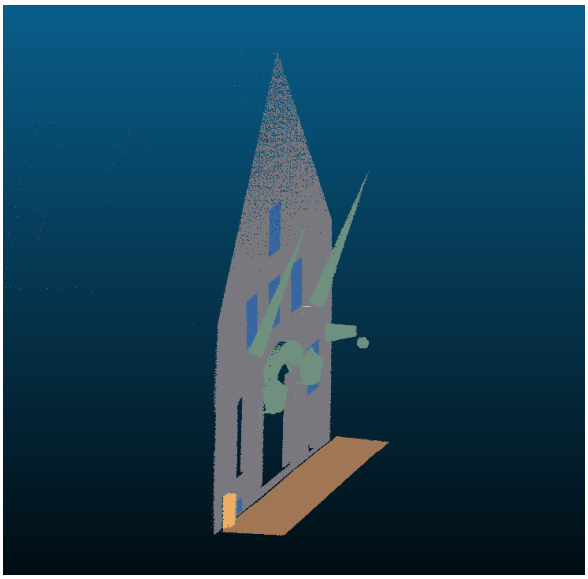
Figure 20: The images are synthetic point clouds sampled from a façade in real MLS data. The windows (blue region in upper image) on this façde is wrongly segmented as `WallSurface`. The Door (purple region in upper image) on this façde is wrongly segmented as `WallSurface`. Some other vertically flat but non-`WallSurface` surfaces are mis-segmented as well.



(a) True Class Label



(b) Predicted Labels



(c) Side view

Figure 21: The images are synthetic point clouds sampled from a façade in LOD3 building data. The windows (blue region in upper image) on this façde is wrongly segmented as `WallSurface`. The window points inference from LOD3 buildings constructed structure-less surfaces that blend into the walls. As one can observe from the side view, the front surface of façade is a completely flat surface without any structure.

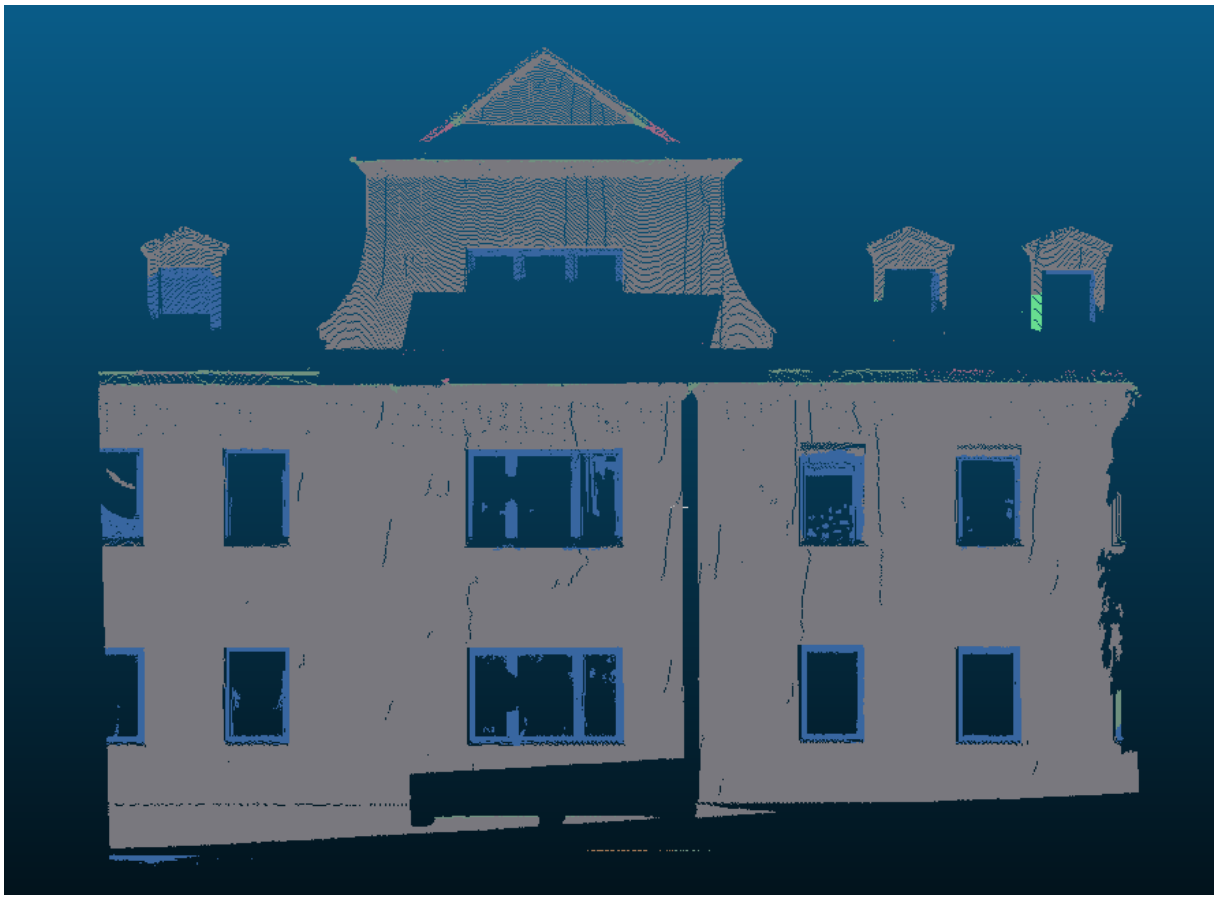


Figure 22: This image presents the prediction results of a façade in real MLS data. The `BuildingInstallation` objects on the top this façde are wrongly segmented as `WallSurface`, as the main components of these installations are vertical planar surfaces.