



TECHNICAL UNIVERSITY OF  
MUNICH

TUM Data Innovation Lab

3D Matching of TerraSAR-X Derived  
Ground Control Points to Mobile Mapping  
Data

Authors	Louise Loesch, Islam Mansour, Magdalena Reich, Xianbin Xie
Mentor(s)	Dr. Wolfgang Koppe, Tatjana Bürgmann (Airbus Defence and Space GmbH)
Co-Mentor	Laure Vuaille
Project lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

## Abstract

In this report, we focus on the automatic extraction of pole-like structures from mobile LiDAR point clouds which can be used for 3D matching with Ground Control Points derived from TerraSAR-X satellite data. Different approaches were analyzed for point cloud classification to extract pole-like structures from the LiDAR data. We present two different frameworks for the point cloud classification. The first approach is based on classic machine learning after applying feature extraction for each 3D point with respect to its neighbors, after which each 3D point is labelled with a semantic class. The second framework is mainly based on a deep learning approach for point cloud classification. In both frameworks, the provided datasets were used for training models. Afterwards, the 3D points classified as light poles are used for the extraction of Pole Control Points (PCPs) that mark the ground point of a pole. At last, the PCPs are used for 3D matching with the ground control points from TerraSAR-X. For this matching, we present an iterative algorithm based on state-of-the-art approaches for 3D point set registration that can handle different degrees of coordinate distortion in the PCPs.

## Contents

Abstract .....	2
Contents.....	3
List of Figures.....	5
List of Tables .....	6
1. Introduction.....	7
2. Problem Definition and Project Plan.....	8
2.1. Datasets .....	8
2.1.1. Point Cloud.....	8
2.1.2. Ground Control Points .....	9
2.2. Segmentation of Mobile Mapping Data.....	10
2.3. Matching of Mobile Data and Satellite-Derived Ground Control Points .....	10
3. Literature Research.....	12
3.1. Segmentation .....	12
3.2. Classification .....	13
3.3. 3D Point Set Registration.....	13
4. Approaches .....	15
4.1. Segmentation and Classification.....	15
4.1.1. Classical Machine Learning .....	15
4.1.2. Deep Learning .....	18
4.1.3. Results .....	19
4.2. Extraction of light for matching .....	22
4.3. Matching.....	23
4.3.1. Selected Algorithms.....	23
4.3.2. Implementation and Testing.....	24
4.3.3. Results .....	25
4.4. Software Prototype.....	27
5. Conclusion .....	28
Bibliography .....	29
Appendix .....	31
Appendix A: Gitlab Repository and Responsibilities .....	31
Appendix B: Comparison of Point Cloud file format .....	34
Appendix C: Point Clouds Features Definition .....	35

Appendix D: Additional Data from Matching .....	36
Appendix E: Software Documentation.....	37
Appendix F: Potential improvement of different methods .....	43

## List of Figures

Figure 1: Dataset of Werk2_Classified_part1 point cloud .....	9
Figure 2: Ground Control Points overlaid above optical satellite image .....	10
Figure 3: Synthetic representation of the segmentation methods [2]. .....	12
Figure 4: Overview of the proposed pipeline of our approaches .....	15
Figure 5: Overview of the proposed methodology.....	16
Figure 6: Left: Point cloud in 3D, Right: Point cloud projected to 2D Plane.....	17
Figure 7: Comparison between the data used in PointSeg and our data .....	18
Figure 8: Preprocessing of our data for PointSeg algorithm .....	18
Figure 9: Comparison of prediction with PointSeg between their data and our data.....	18
Figure 10: Left: Model trained with 22 3D features, Right: Model trained with 26 3D and 2D Features.....	19
Figure 11: Left: Confusion Matrix of the testing set of “Werk2_classified_part1”, Right: visualization .....	20
Figure 12: Left: Confusion Matrix of the testing set of “086b_classified”, and the visualization .....	20
Figure 13. Points assigned to wrong classes are shown in blue.....	21
Figure 14: Qualitative results with PointNet++ .....	21
Figure 15: Confusion matrix for deep learning.....	22
Figure 16: DBScan Clustering of PCPs.....	23
Figure 17: Demonstration of GLMDTPS. ....	24

## List of Tables

Table 1: Dataset of point cloud and its number of points.....	9
Table 2: List of GCP Datasets.....	10
Table 3: List of defined features .....	16
Table 4: Results of iterated GLMDTPS with different thresholds for nearest neighbor matching .....	25
Table 5: Distances of matches before and after iterated GLMDTPS .....	26
Table 6: Distances between matches of artificially distorted data before iterated GLMDTPS and after.. .....	26

## 1. Introduction

Autonomous driving is a field that has been gaining a lot of interest over the past few decades. A self-driving car needs to have precise information and instructions about its location in context to its surroundings to safely navigate in the 3D world. For this purpose, so-called High Definition maps or HD maps are created that have extremely high precision down to centimetre-level.

Usually, a combination of different sensors such as radar and optical camera, GPS, and LiDAR, short for Light Detection and Ranging, which is a remote sensing method that uses a pulsed laser to measure ranges, are used to create HD maps and then navigate the car. This combination of different information provides a certain redundancy so that, in case some of the sensors fail to work, the remaining information is enough to navigate.

However, the location provided by GPS sensors does not result in satisfactory levels of precision in many cases and can be highly distorted in case of complete or partial signal loss during the acquisition process, which means that the scans provided by the other sensors are associated with inaccurate coordinates.

As a solution to this problem, Airbus Defence and Space proposes using geolocation data gained from the radar satellite TerraSAR-X to correct the inaccurate GPS locations associated with the data from other sensors in order to create a highly accurate HD map, or to validate existing maps for increasing reliability.

Synthetic-aperture radar, such as the satellite uses to acquire images and Digital Elevation models, is capable of obtaining two-dimensional images or three-dimensional reconstructions of objects from orbit with a very high absolute geolocational accuracy. From these SAR images, Ground Control Points (GCPs) can be accurately extracted. GCPs are generally defined as points on the surface of the earth of which the coordinates are precisely known, that are used to geo-reference satellite imagery, and in this case mark the location of metallic pole-like structures on the ground or on top of buildings, since these poles have high backscatter in the radar images and can therefore easily be identified [1]. Based on this approach, a dense point cloud of GCPs can be gained with a geo-positional accuracy of less than ten centimetres.

In order to use the accuracy of these GCPs for autonomous driving, the goal of our project was to detect pole-like structures in a point cloud gained from LiDAR, then compute the coordinates of the base point of these structures, and finally match them with the GCPs. Based on the matching result, we compute a transformation function to correct the coordinates of the data to match the GCPs.

## 2. Problem Definition and Project Plan

The main objective of the project is to extract the ground points of pole-like structures from mobile LiDAR data and match these points with the corresponding GCPs, which are derived from TerraSAR-X satellite data. The project was divided into different work packages for each team member to work on specific tasks. The team consists of four students with different backgrounds and study majors: mathematics, urban planning, aerospace and geodesy, and information technology. A link to the full implementation of our results and an explanation on which student was responsible for which part can be found in Appendix A.

The main tasks, as shown in Appendix A, Figure 1, for the project were defined as follows:

1. Literature Review
2. Data Exploration & Initial Analysis
3. GCPs Matching Algorithms
4. LiDAR Segmentation & Classification
5. The fusion of LiDAR Segmentation & Classification and GCPs Matching
6. Prototype for LiDAR

The main project objective was divided into different sections. The first is to semantically segment and classify a point cloud which is obtained from mobile LiDAR scans of a street and extract light poles, of which then the ground points are extracted. In the following, these ground points are referred to as Pole Control Points (PCPs). The second is to find matching GCPs and PCPs. Based on this matching, a transformation matrix can be calculated, which is required for geometric correction of the point cloud based on the high accuracy position of GCPs. Therefore, the mobile LiDAR data can be corrected to obtain high definition maps of the surroundings.

### 2.1. Datasets

Airbus Defence and Space provided three different datasets of mobile LiDAR data showing different sceneries in Friedrichshafen, two datasets of Ground Control Points and satellite images for visualization, which are listed below:

1. GCPs (Shapefile)
2. Satellite imagery for the given location (Geotiff)
3. Mobile LiDAR datasets for different sceneries (LAS, BIN (CloudCompare))

#### 2.1.1. Point Cloud

The cloud point is a set of data points of scanned objects in a three-dimensional space. In our case, the point cloud was generated from a LiDAR mobile scanner and provided to us for analysis as listed in Table 1. There is a variety of file formats for storing LiDAR data and each format has its advantages and disadvantages from supporting metadata and projection to adding a new attribute for each point cloud. The classified data was provided as BIN files, which is shown in Figure 1, which is a specific data format for CloudCompare; However,



CloudCompare data format is not supported by any other software for point cloud processing or visualisation. Therefore, the BIN files must be transformed into LAS files which are widely machine-readable, as it is illustrated in Appendix B: Comparison of Point Cloud file format, Table 1, the different supported features for each file format.

Table 1: Dataset of point cloud and its number of points

<i>Dataset Name</i>	<i>Number of points</i>
<i>086b_classified</i>	9,175,355
<i>Werk2_classified_part1</i>	2,044,148
<i>Werk2_classified_part2</i>	22,043,528

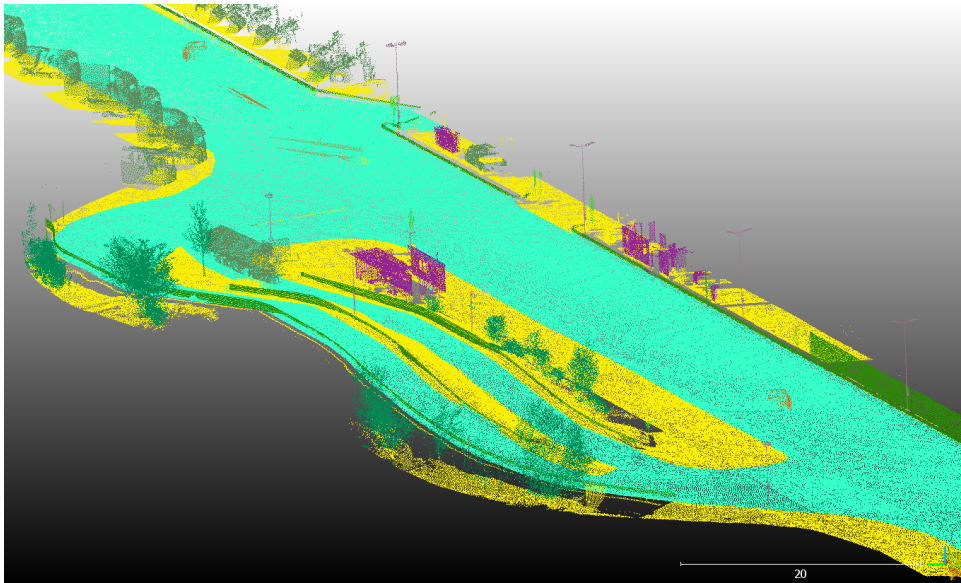
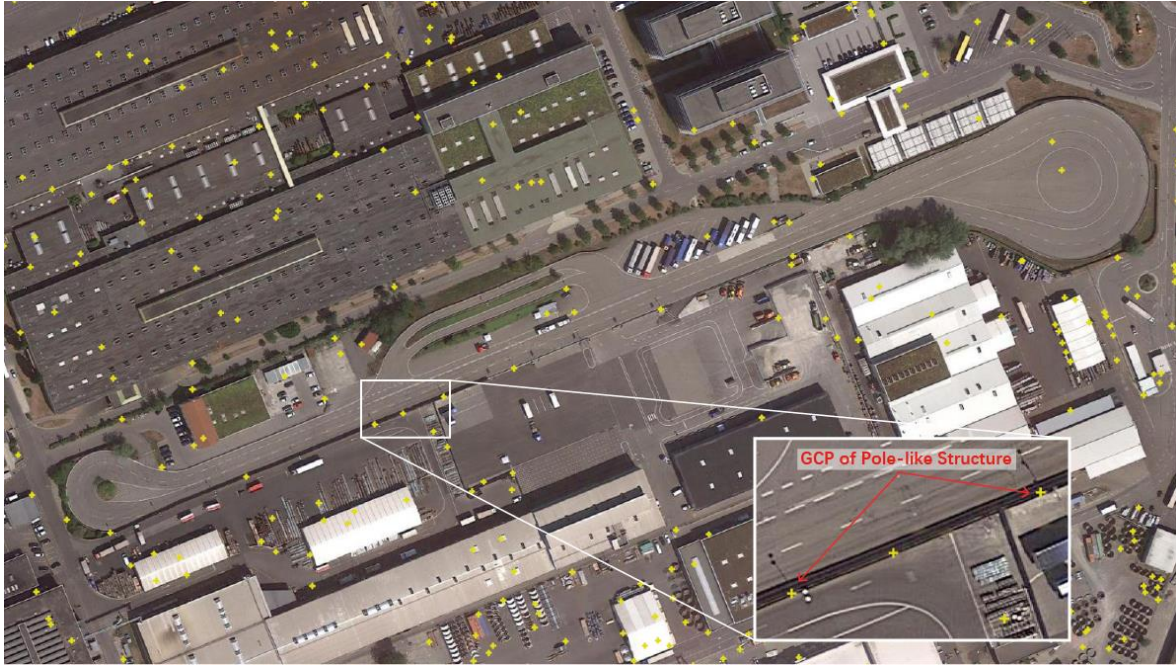


Figure 1: Dataset of Werk2\_Classified\_part1 point cloud

### 2.1.2. Ground Control Points

The Ground Control Points provide the exact location of metallic, pole-like structures identified from synthetic aperture radar scans supplied by the TerraSAR-X satellite, as illustrated in Figure 2. These datasets do not only contain poles along streets but also other metallic objects which appear brighter on the images due to the radar signal reflection properties of these objects, for example an antenna on a rooftop or other metallic structures.



**Figure 2: Ground Control Points overlaid above optical satellite image**

We were provided with two independently computed sets of GCPs of the area Friedrichshafen, of which both contain GCPs that are not represented in the other set.

Each dataset contains a list of GCPs with ID and 3-dimensional coordinates in  $xyz$  format.

**Table 2: List of GCP Datasets**

<i>GCP Set Name</i>	<i>Number of GCPs</i>
<i>GCP set 1</i>	6912
<i>GCP set 2</i>	6185

## 2.2. Segmentation of Mobile Mapping Data

More and more 3D models are available, but the methods to retrieve information from them and give meaning to the data are still in development. Segmentation is one way to analyze this data. Traditionally, the idea of segmentation of 3D point cloud comes from the segmentation of the 2D image, for example by registering each point into a voxel or projecting the 3D object into 2D scenery. However, the algorithms for segmentation are computationally expensive when it requires preprocessing like voxelization or 2D projection and thus take a long time to be computed. Furthermore, both voxelization and 2D projection result in information loss and worse performance for the detection of small objects. Therefore, our aim is to segment the point cloud directly without voxelization or rendering the point cloud in 2D and at the same time to achieve high accuracy.

## 2.3. Matching of Mobile Data and Satellite-Derived Ground Control Points

The second step, the matching of the poles from the mobile mapping data and the GCPs from the satellite images, is based on two datasets. One contains the ground coordinates of the poles

along a street (PCPs), that have been extracted based on the classification from the first step, while the other contains the accurate GCP coordinates for a much larger area. Both datasets are projected into the coordinate reference system UTM32N.

The importance of this step lies in the fact that the coordinates of the mobile mapping data are slightly inaccurate in almost all cases and might be highly distorted due to a loss of GPS signal during the acquisition process in some cases. In order to use the acquired mobile mapping data for autonomous driving, it is necessary to correct it to match the accurate GCPs.

Therefore, in this step, the goal is to find two unknown variables. First, we identify correct matches between the two datasets with a low rate of false positives, and then use these matches to compute a transformation that can be used to correct the coordinates of all points from the LiDAR scan data.

The problem posed by the matching between PCPs and GCPs is in general terms an instance of point set registration, which is defined as follows: Given two sets of points in an  $n$ -dimensional vector space, find a transformation that best aligns the *source set* to the *target set*. In our case, the source set corresponds to the dataset of PCPs, the target set is provided by the GCPs, and we are working with three dimensions since we do not have additional characteristics of both GCPs and PCPs that can be used for matching except for their coordinates in  $xyz$  format.

An additional difficulty is posed by the independence of the two datasets. Since they stem from different scanning technology, are calculated in different ways, and cover a largely different area, there is a very high ratio of both GCPs and PCPs for which no correct match exists.

### 3. Literature Research

The first phase of our project started with an initial literature research to identify possible approaches to solve both parts of the problem, of which we decided on the algorithms that appeared most suitable in our context to implement.

#### 3.1. Segmentation

Segmentation is the attempt to group point clouds into clusters based on different techniques. In literature, there are many approaches to segment 3D models and point clouds: via edge-based segmentation, region growing segmentation, segmentation by model fitting, hybrid segmentation techniques, or machine learning segmentation [2] as shown in Figure 3.

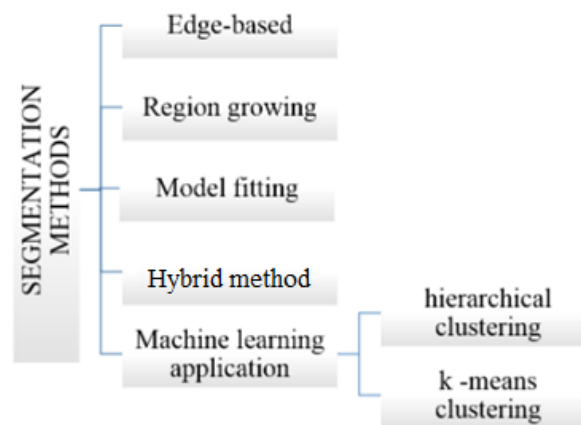


Figure 3: Synthetic representation of the segmentation methods [2].

The traditional approach is based on retrieving features and detect regions from them. One approach is the edge-based segmentation [2]. This approach has two main stages: detection of outline borders of the different regions and then grouping points inside the boundaries to get the final segments. This algorithm allows a fast segmentation but on the other hand, the accuracy drops in case of noise and uneven density of the point cloud.

Another approach using machine learning methods is the region growing segmentation [2] This approach can be implemented using different methods. These methods of segmentation are also in two stages: identification of seed points based on the curvature of each point and growing those seed points based on criteria like proximity or normal. Although these methods are more robust to noise compared to the edge-based segmentation method, they are sensitive to inaccurate estimations of the normal and curvatures of points near region boundaries and the location of initial seed regions.

Besides, deep learning approaches start to develop and the most common network for segmenting a point cloud is PointNet [3]. It is a deep learning framework that directly consumes unordered point sets as inputs and has a unified approach to a number of 3D recognition tasks including object classification, part segmentation and semantic segmentation. The challenges of feeding the raw point cloud into the network are to keep the points unordered, invariant to  $N!$  permutations and to keep the invariance under geometric transformation. PointNet is a

pioneer and has achieved a good result: an overall accuracy of 78.62% on the Stanford 3D semantic parsing data set [4] including 271 rooms and an average IoU of 24.24% which is better than the previous state-of-the-art [4] who achieved 18.22%. It is even robust to data corruption but it does not perform well outdoors as it was trained for indoors scenes and is limited in complex scenes.

In light of these issues, PointSeg [5] is a good solution. It is a real-time end-to-end segmentation method for outdoors scenes with an open-source code. The input must be the points which are projected into spherical coordinates to extract robust feature representation. This architecture has the advantage of being fast and it does not need a lot of memory: The system cost 12 ms per frame in the workstation during the forward process with 2G memory cost. However, it performs badly on small objects: the average IoU of a car is 67.3% but only 23.9% for pedestrians.

Another recently developed deep learning approach is PointNet++ [6]. The input is also the raw point cloud and it partitions the set of points into overlapping local regions by the distance metric of the underlying space. PointNet is applied on each partition to extract local features and then all the features are grouped into larger units and processed to produce higher-level features. This network achieved state-of-the-art performance in a very challenging benchmark: 84.5% accuracy on ScanNet whereas PointNet achieved only 73.9%.

### **3.2. Classification**

As soon as the point cloud is segmented, each segmented group of point can be labelled with a specific class to identify the object and give a semantic meaning to the segment. Recently, the point cloud classification is becoming a very active field of research due to the increasing interest for autonomous driving and another related field. The class labelling for each point cloud is achieved following three different approaches [2]. The first is a *supervised approach*, where semantic categories are learned from a dataset of annotated data and the trained model is used to provide a semantic classification of the entire dataset. A large amount of annotated data is normally mandatory to train the model. A second approach is an *unsupervised approach*, where the data is automatically partitioned into segments based on a user-provided parameterization of the algorithm. No annotations are requested but the results might not correspond to user-defined classes with a specific semantic meaning. The last approach is an *interactive approach*, where the user is actively involved in the segmentation/classification loop by guiding the extraction of segments via feedback signals. This requires a large effort from the user side but it could adapt and improve the segmentation result based on the user's feedback.

### **3.3. 3D Point Set Registration**

In the 3D case, point set registration is most often used to align two scans of the same object or scenery or find the correct position of a small part of the scenery in the complete picture [7]. While this last use case seems similar to our problem setting, the matching approaches designed for that scenario assume complete scans of the sceneries to be matched, and therefore are based either on local features like curvature that rely on a high density scan of a surface to be

meaningful [8] or use a deep learning approach that requires a large amount of data [7], while our point sets are very sparse and lack meaningful local features. Hence, we decided to focus on more general approaches.

We can differentiate between two different types of point set registration: rigid and non-rigid registration. In the case of rigid point set registration, it is assumed that the transformation needed to transform the source set into the target set is rigid, which means that it does not change the distance between two points, or equivalently, that the transformation is composed purely of rotation and translation [9]. On the other hand, in non-rigid point set registration, the transformation may be affine or otherwise non-linear [10].

In the following, we will give an overview of some state-of-the-art approaches in point set registration.

The most well-known algorithm in the field of rigid point set registration is the *Iterative Closest Point* algorithm (ICP) [9]. Assuming that the two-point sets are roughly aligned, it iterates steps of computing a match of each point in the source set to the nearest point in the target set and computing a rigid transformation that minimizes a chosen distance metric, originally a mean squares metric. In its most basic form, ICP has been shown to monotonically converge to the nearest local minimum. There exist many variants that address different weaknesses, such as sensitivity to noise or outliers [11] or to ensure a more globally optimal minimum [12].

Another prominent example of a rigid point set registration is *Robust Point Matching* (RPM) [10]. While it is able to find affine transformations in the 2D case, the 3D case only considers rotation and translation. RPM searches for a one-to-one correspondence between the point sets via a soft-assign approach. This means that instead of a binary correspondence, where the value 1 stands for a match between two points and value 0 signifies no match, each pair of points gets assigned an initial value between 0 and 1 signifying a sort of degree of the match, and these values are iteratively updated before ultimately converging to either 0 or 1 while filling the two-way assignment constraints necessary to guarantee the one-to-one matching. RPM has been shown to be robust to noise and outliers due to its use of slack parameters.

Based on RPM, a very notable approach to non-rigid point set registration is the TPS-RPM [13], which uses the same soft-assign method, but parametrizes the transformation with the help of *thin-plate splines* (TPS), a technique for interpolation based on splines with a constraint on smoothness. In general, without the rigidity constraint, it is possible to map any point of the source set to any arbitrary point of the target set. To control the arbitrariness of the mapping, TPS-RPM uses the smoothness constraint of the TPS. Thin plate splines are the most common approach to model non-rigid transformations.

Another approach that uses TPS is the GLMDTPS algorithm, short for Global and Local Mixture Distance TPS [14]. It starts by computing a more rigid matching based on a global distance metric and then iteratively moves towards a non-rigid matching weighing local distances more heavily by using an annealing scheme. GLMDTPS shows impressive results on par to several state-of-the-art algorithms including TPS-RPM and a probabilistic method called GMMREG in trials on cases with high outlier ratio, noise, or partially missing data.

## 4. Approaches

The overall pipeline for our approaches is shown in the following Figure 5. Segmentation and classification will be approached by both machine learning and deep learning. Next, the best classification result is chosen, and the PCPs are extracted from that result. Finally, the PCPs are matched to GCPs. Each part mentioned above will be discussed in the following subsections.

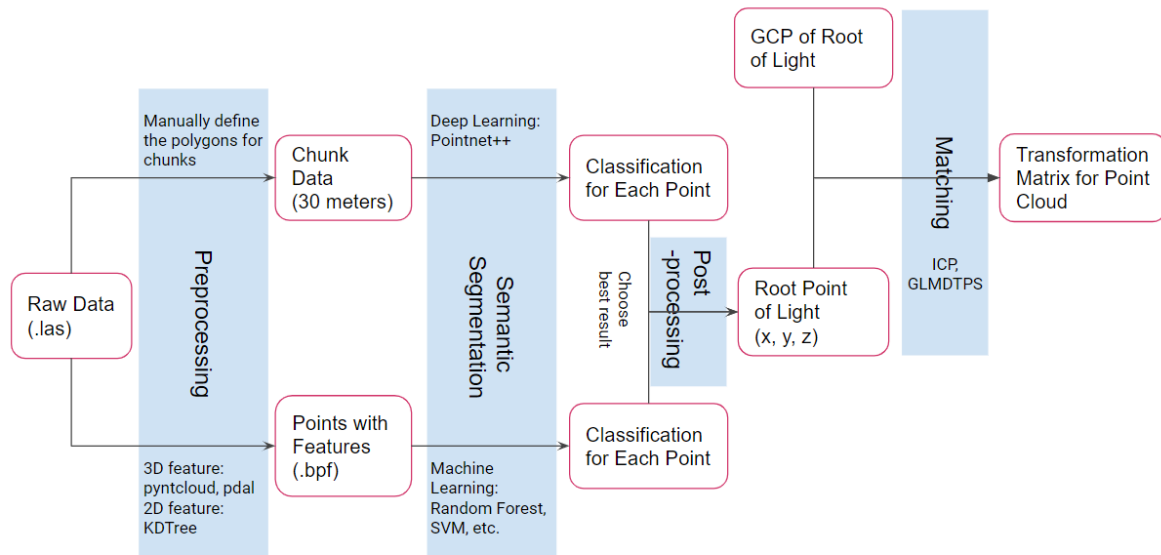


Figure 4: Overview of the proposed pipeline of our approaches

### 4.1. Segmentation and Classification

In this project, the segmentation of the point cloud is approached based on two different methods, namely classical machine learning and deep learning. Each method is benchmarked to investigate its advantages and disadvantages in the context of the provided problem.

In the following subsection, we briefly summarize the main pipelines for point cloud classification based on the two approaches: machine learning with extracted features of each point and deep learning with only geospatial information.

#### 4.1.1. Classical Machine Learning

In general, classic machine learning (random forest, support vector machine, etc...) and feature extraction are applied for as an approach for point clouds classification. The basic method is adapted from [15] and is shown in Figure 5. However, it has been applied differently to suit our needs and expected results. In addition, another list of features is added to increase the accuracy of the classifier. Mainly, we have added 2-D Eigenvalues, 2-D density, and height of each point relative to the lowest point in the point clouds.

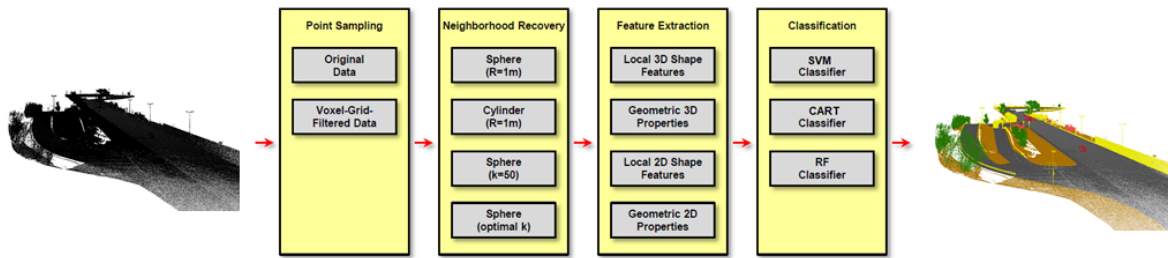


Figure 5: Overview of the proposed methodology

First, we composed operations on point clouds into a pipeline to request PDAL, a library used to process point cloud data. PDAL is based on C/C++, therefore the process is much faster than python. In our experiment, we used PDAL to create the first 14 features in the Table-1 for the dataset with 9175355 points and it took about 72 minutes only, meanwhile, it was not possible to compute any of the features in python because of the iPython kernel crashes due to encountered out of memory. In this process, a regional point cloud is defined by each point in the whole dataset and its 8 nearest neighbours. And their eigenvalues, eigenvectors, rand and so on were computed in regard to each regional point cloud.

Second, the features describing the geometrical form of local point clouds were computed as DataFrame by Pandas. Eigenvalues devived from the first step were used here to compute the new features.

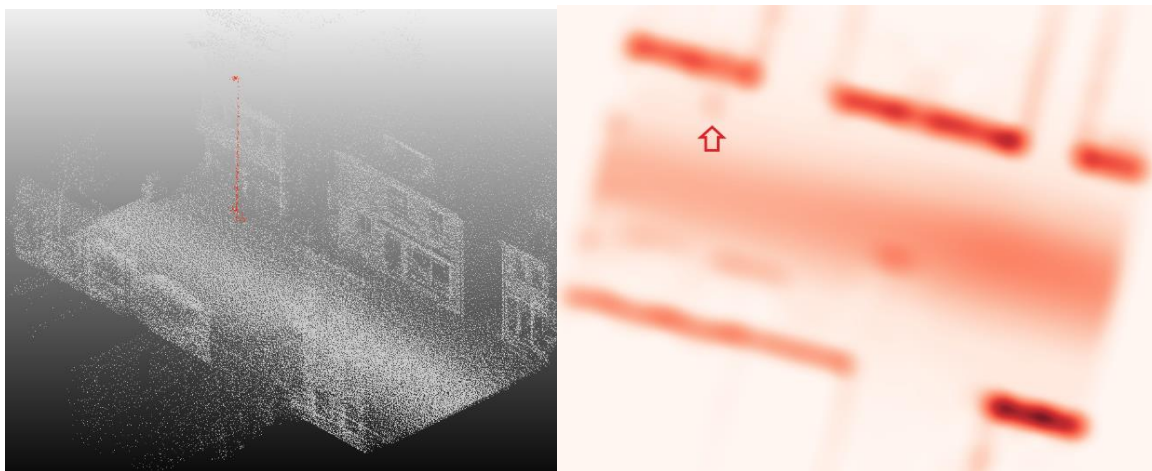
Finally, by experiment, we found that the 2D features and Z coordinates of the point cloud could also contribute to a better classification accuracy (experiment result can be found in 4.1.3.1). It is intuitively understandable because, in the urbanized area, many artificial structures are vertically distributed, such as pole, signal, and facade of a building. As shown in Figure 6, when the point clouds of this structure are projected onto the 2D plane, their density will be much higher than other objects. What's more, many objects, for example, tree and vehicle, are located in different heights. Therefore, the Z coordinate should also benefit the classification of points cloud, while in our trial the machine learning method could hardly learn from X and Y coordinates. Based on these findings, we provided four more features of each point for the following classification process, namely the 2D density of the regional points within a given radius (here the radius is set to 20 cm), its 2 eigenvalues, and the Z coordinate as defined in Table 3 and Appendix C: Point Clouds Features Definition.

Table 3: List of defined features

	<i>Feature Name</i>	<i>Description of Feature</i>
1	<i>KDistance</i>	The Euclidean distance to a point's 8-th nearest neighbour
2	<i>LocalReachabilityDistance</i>	The inverse of the mean of all reachability distances for a neighbourhood of points
3	<i>LocalOutlierFactor</i>	The mean of all LocalReachabilityDistance values for the neighbourhood
4	<i>NNDistance</i>	Similar to KDistance
5	<i>Eigenvalue2</i>	The largest Eigenvalue based on its 8-nearest neighbours in 3D.



	<i>Feature Name</i>	<i>Description of Feature</i>
6	<i>Eigenvalue1</i>	The second-largest Eigenvalue based on its 8-nearest neighbours in 3D.
7	<i>Eigenvalue0</i>	The smallest Eigenvalue based on its 8-nearest neighbors in 3D.
8	<i>Rank</i>	Computed by SVD with 8-nearest neighbours. Point sets with rank 1 correspond to linear features, while rank 2 correspond to planar features and rank 3 corresponds to a full 3D feature.
9	<i>NormalX</i>	The normal is taken as the eigenvector corresponding to the smallest eigenvalue.
10	<i>NormalY</i>	
11	<i>NormalZ</i>	
12	<i>Curvature</i>	Smallest eigenvalue divided by the sum of all three eigenvalues.
13	<i>RadialDensity</i>	The density of points in a sphere of a given radius. Here the radius is 2.
14	<i>Coplanar</i>	Technique to performs a fast and robust octree-based segmentation of approximately coplanar clusters of samples. [16]
15	<i>Linearity</i>	$(\text{Eigenvalue0} - \text{Eigenvalue1}) / \text{Eigenvalue2}$
16	<i>Planarity</i>	$(\text{Eigenvalue1} - \text{Eigenvalue0}) / \text{Eigenvalue2}$
17	<i>Scattering</i>	$\text{Eigenvalue0} / \text{Eigenvalue2}$
18	<i>Omnivariance</i>	$(\text{Eigenvalue0} * \text{Eigenvalue1} * \text{Eigenvalue2})^{**}(1/3)$
19	<i>Anisotropy</i>	$(\text{Eigenvalue2} - \text{Eigenvalue0}) / \text{Eigenvalue2}$
20	<i>Eigentropy</i>	$-(\text{Eigenvalue0} * \log(\text{Eigenvalue0}) - \text{Eigenvalue1} * \log(\text{Eigenvalue1}) - \text{Eigenvalue2} * \log(\text{Eigenvalue2}))$
21	<i>Eigen_Sum</i>	The sum of all three eigenvalues
22	<i>Curvature_Change</i>	$\text{Eigenvalue0} / (\text{Eigenvalue0} + \text{Eigenvalue1} + \text{Eigenvalue2})$
23	<i>density_2d</i>	The density of points, which are projected to X-Y plane, in a circle of a given radius. Here the radius is 20 cm.
24	<i>e1_2d</i>	The smallest Eigenvalue based on its neighbours within 20 cm in 2D.
25	<i>e2_2d</i>	The largest Eigenvalue based on its neighbours within 20 cm in 2D.
26	<i>Z</i>	Coordinate in Z-axis.



**Figure 6: Left: Point cloud in 3D, Right: Point cloud projected to 2D Plane**

For the multi-classification task, we have chosen RandomForest as a classifier with respect to both accuracy and efficiency. In our experiment, we always took 80% of the whole point cloud

as a training set and the other 20% as the testing set. It took only 2 minutes 26 seconds to train a RandomForest model for a point cloud with 2044148 records in total, and 17 minutes 49 seconds for another point cloud with 9175355 records. SVM was also considered as a classifier, but it has been proven infeasible as the training with approximately 1 million records took 6 days and did not finish. The detailed performance of classification with RandomForest will be discussed in chapter 4.1.3.

### 4.1.2. Deep Learning

The second approach using Deep Learning, we first tried to use the public code of PointSeg. As preprocessing, it is needed to project the point cloud onto a spherical coordinates system. However, our data was very different than the data used by the developers: We used a full scenery of a street, while they used a panoramic vision of an embedded sensor as it is used for real-time autonomous driving as shown in Figure 7.



Figure 7: Comparison between the data used in PointSeg and our data

In order to adapt our data for the algorithm, we created chunks from the entire scenery, and each chunk was centred. Then we projected each chunk onto a sphere as shown in Figure 8.

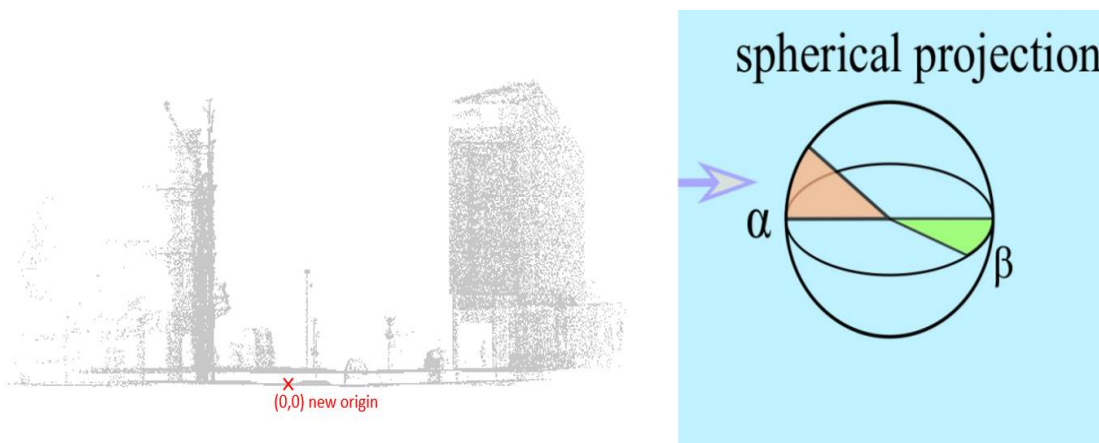


Figure 8: Preprocessing of our data for PointSeg algorithm

Even though we tried to adapt our data in the same way as the data used for training the PointSeg architecture, we never managed to make the algorithm work successfully. The results we achieved were poor as can be seen in Figure 9.

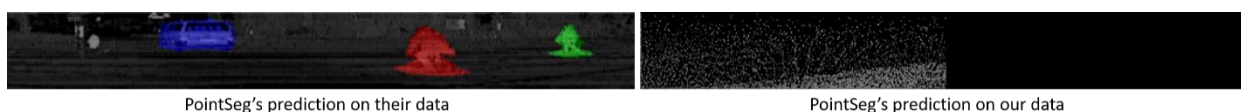


Figure 9: Comparison of prediction with PointSeg between their data and our data

Because it was difficult to adapt our data for the PointSeg algorithm, and in addition to that, because it was stated in [5] that the performance of detection of small objects was not promising due to the downsampling by pooling layers in the architecture, we decided not to continue this path. Since our final goal was to extract the pole-like structures which are comparably small objects, we tested another algorithm: PointNet++.

PointNet++ uses Open3D and TensorFlow GPU as framework. We needed to train the network with our data in order to predict light structures as it was not a class that the architecture was trained to recognize. Nevertheless, we did not have much data for training and testing, only three sceneries, and each data was labelled differently. So we used the chunks of 30m\_x\_30m created for PointSeg. We divided the chunks into a training set, validation set and test set, and trained the model for 500 epochs.

### 4.1.3. Results

#### 4.1.3.1. Random Forest

For the task of multiclass-classification with classic machine learning, we chose RandomForest as classifier because of its high accuracy and efficiency. In every experiment, we split our dataset into a training set (80%) and testing set (20%).

In the first experiment, we used the dataset “Werk2\_classified\_part1” with 2044148 records to compare the models with or without 2D features. The results are shown in Figure 10 and Figure 11. It is clear that 2D features contribute to increasing the prediction precision for each class. Especially the precision of building is improved by 31%, which corresponds to our assumption that the vertically distributed structure would benefit from the 2D features mostly among all classes. The precision of light has also risen significantly from 78% to 91%, while the overall accuracy changed from 83.58% to 91.45%.

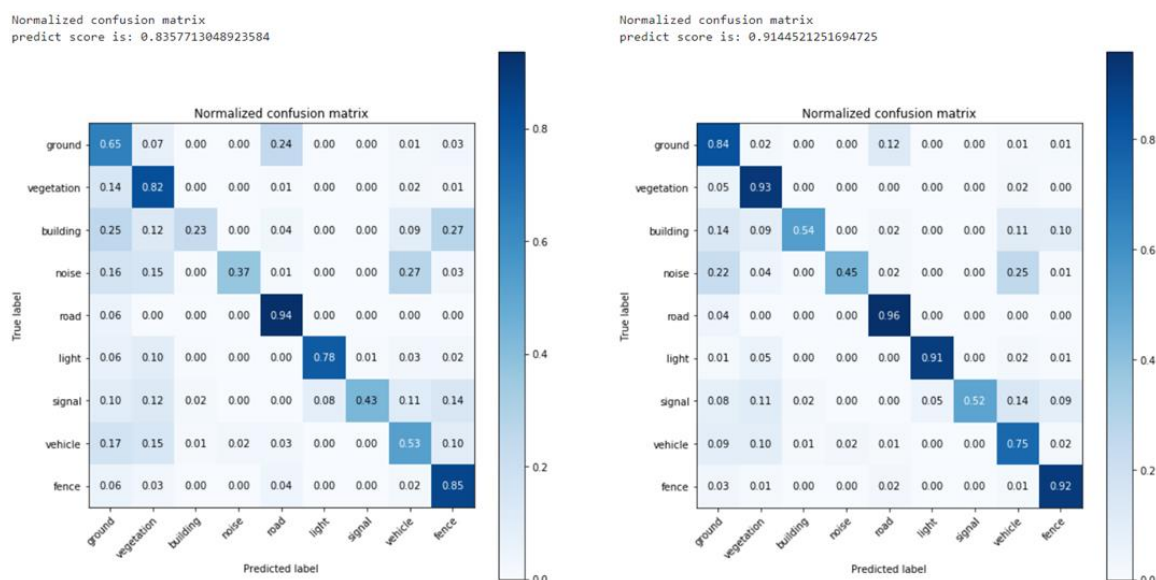


Figure 10: Left: Model trained with 22 3D features, Right: Model trained with 26 3D and 2D Features

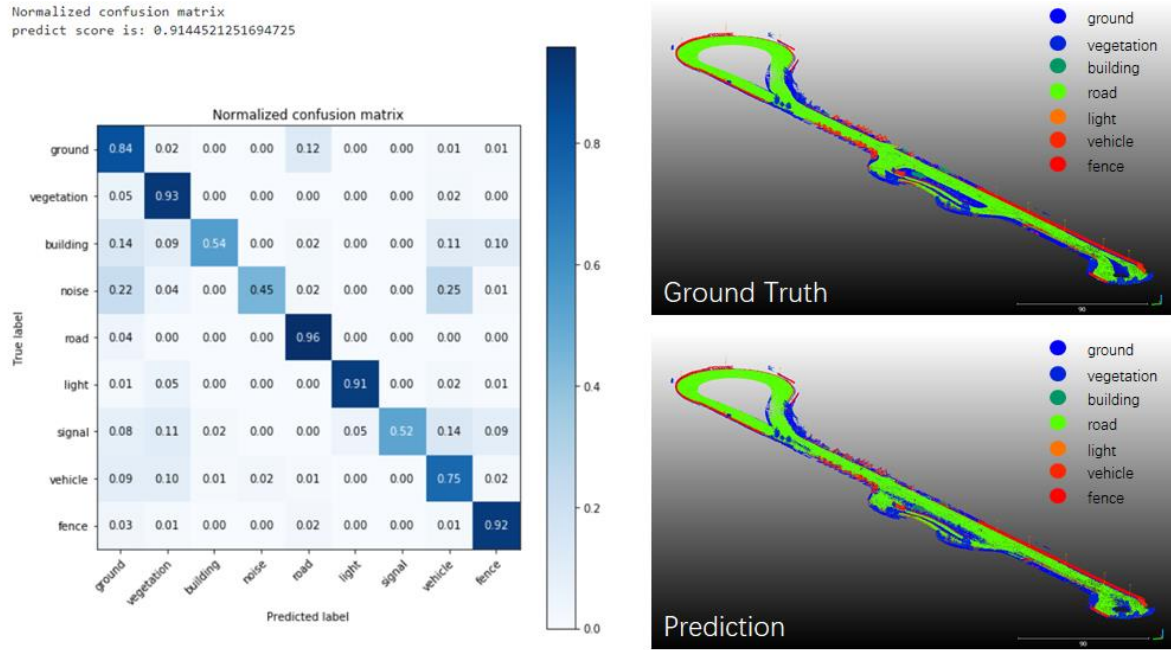


Figure 11: Left: Confusion Matrix of the testing set of “Werk2\_classified\_part1”, Right: visualization

For the dataset “086b\_classified”, we used the 26 features to train the RandomForest model. We achieved an overall accuracy of 85.82% and a precision for the class “light” of 63%, which were lower than the results for dataset “Werk2\_classified\_part1”, as shown in Figure 11.

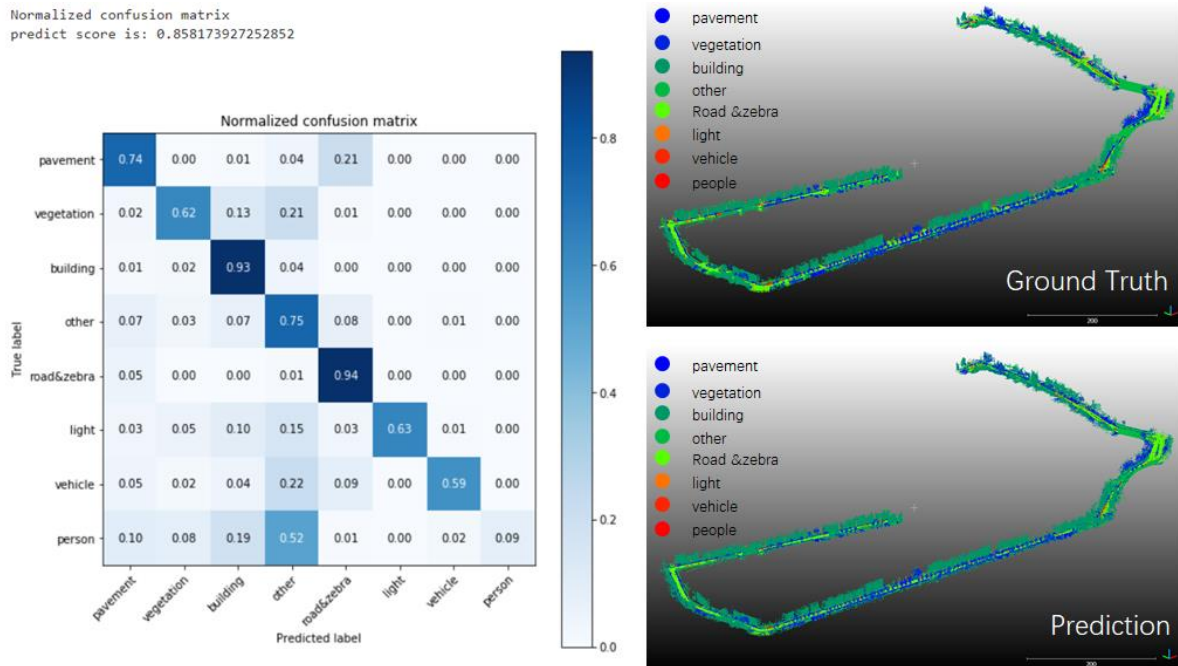


Figure 12: Left: Confusion Matrix of the testing set of “086b\_classified”, and the visualization

By looking closer at the confusion matrix, we can find that many points of pavement are labelled as a road. In Figure 12 and Figure 13, the points predicted as wrong classes are marked in blue, and it can be shown that many errors occur for “pavement”. Furthermore, many points

of “vegetation” and “light” are predicted as “other” or “building”, which reveals that the urbanized environment of “086b\_classified” is more complicated than the “Werk2\_classified\_part1”, because in the “086b\_classified” there are many trees along the street which may hide lights and buildings in their branches. In the later dataset, there are fewer trees and buildings, therefore the light poles are more outstanding from the context.

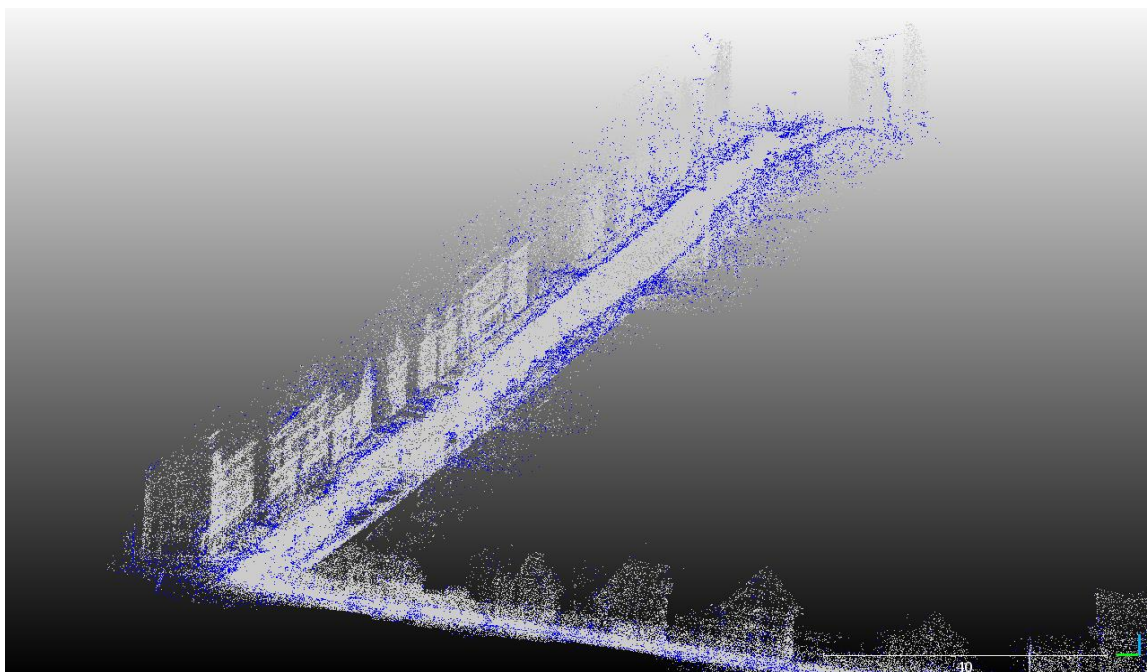


Figure 13. Points assigned to wrong classes are shown in blue

#### 4.1.3.2. PointNet++

To evaluate our results with PointNet++, we used the Intersection over Union (IoU) as a metric. The overall accuracy is 87% and the average IoU (ignoring label ‘other’) is 47% (cf Figure 14). We computed the confusion matrix to help us understand where the errors come from.

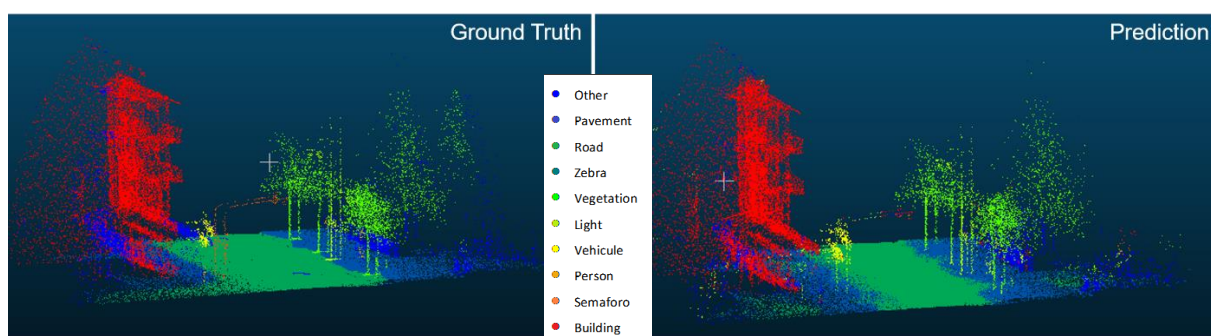


Figure 14: Qualitative results with PointNet++

We can see from the confusion matrix below Figure 15 that the network predicts semaforo (traffic light) as light, which is understandable as both have a pole-light structure, and zebras as road, also understandable as zebras are part of the road Figure 15. However, the network has difficulties recognizing persons, it predicts them as noise. Furthermore, the algorithm is trained on a chunk of 30mx30m, so it will only give results on a similar size of chunks.

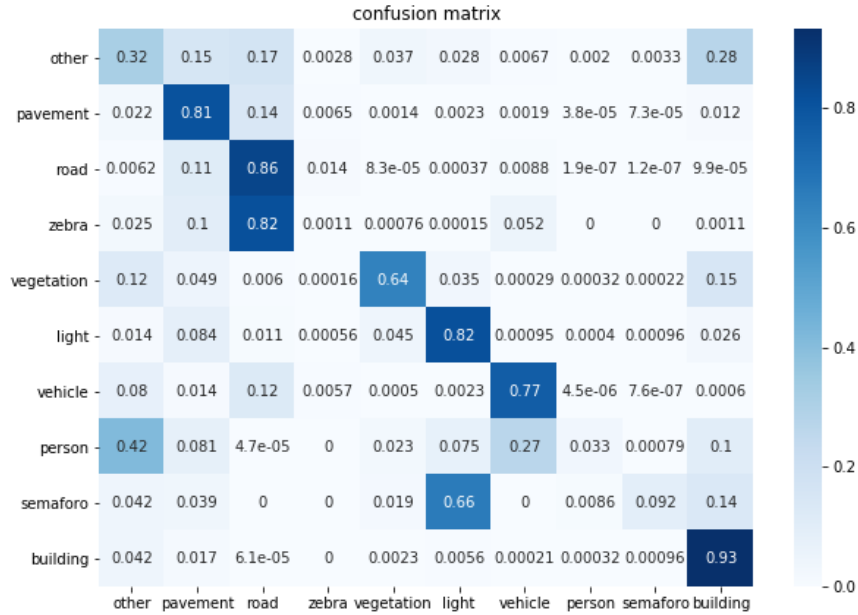


Figure 15: Confusion matrix for deep learning

#### 4.1.3.3. Discussion

The Random Forest algorithm is robust to the size of the input as long as the input is at least 1m long, whereas the PointNet++ algorithm only works on small input, here 30m\_x\_30m. That is why we used the Random Forest algorithm on the three sceneries that was provided to us for the next steps of the project.

However, the Random Forest algorithm is working with features extracted from the point cloud, and that takes a long time to extract especially when the point cloud is big. Indeed, for the Random Forest to be robust and accurate, it requires a large number of features for each point. As an example, it took us 1 hour and a half to extract features for a point cloud of 9 million points and the prediction is still to be done. In comparison, the PointNet++ algorithm is quite fast to compute as it takes the point cloud directly as input. The prediction of the scenery of 9 million points cut into 74 chunks is done in almost 15 minutes.

Both our algorithms for segmentation and classification are trained in an urban area in Germany. It means that they are probably not robust to a different type of data, like a countryside area or even an Asian area.

## 4.2. Extraction of light for matching

Once the point cloud is segmented, we want to extract each individual light structure and compute their PCP, in order to provide source dataset for the matching with GCPs.

To extract each individual light structure, we need to find clusters among the points predicted as “light”. Several algorithms can find clusters from a point cloud, including K-Means, the most famous one, and DBScan.

Using K-means for clustering requires a priori parameter K. But in reality, we usually do not know how to choose K because the number of light varies a lot among scenes. To solve this problem, we tested K-Means on our prediction by iterating a range of different value of K and

selecting the one with the highest IoU. This method only works if we have ground truth. However, in most practical cases the ground truth is not accessible.

Instead, we can use DBScan because it takes the distance between points  $\epsilon$  and the minimum number of points in each cluster as parameters. After tuning of those parameters on the 3 sceneries, we found a default setting:  $\epsilon=0.003$  and  $\text{min\_samples}=10$ . See results below for two different datasets after the random forest classification in Figure 16.

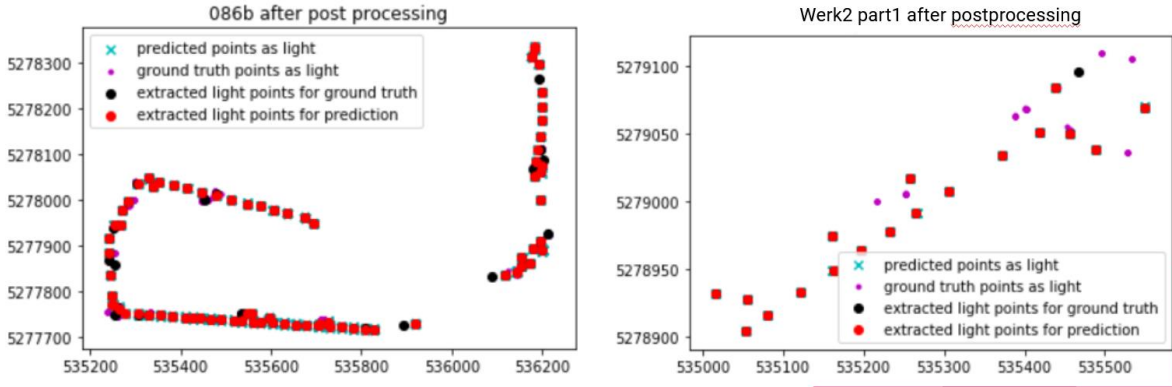


Figure 16: DBScan Clustering of PCPs

The fine-tuning accomplished is for a sparse point cloud. If the point cloud is dense, the clustering will be wrong. A possibility of improvement is to make the parameters of DBSCAN proportionate of the density, but it is still an empirical approach. A second possible way to improve the result is to look into machine learning or deep learning approaches for clustering, even maybe find or create an architecture that can segment and classify a point cloud and extract light structures.

### 4.3. Matching

For the matching, we started working with the already classified “Werk2\_classified\_part1” data and the “GCP set 1” set and later proceeded to test our algorithm with the results from the classification on the “086b” data and the “GCP set 2” set.

#### 4.3.1. Selected Algorithms

The matching occurs in two different types of scenarios. In most cases, if the GPS was nearly accurate during acquisition, a rigid matching is sufficient to provide us with correct matching results, since the GPS inaccuracy can be interpreted as a slight noise on the source set. We operated under the assumption that in this case, the inaccuracy of the mobile coordinates was not more than 100 cm, which is in accordance to the information provided by Airbus. Based on our initial literature research, we decided to take ICP into consideration to solve these application cases for its computational simplicity and because the outlier ratio and minor noise do not pose a high danger of mismatching due to the fact that both datasets are very sparse, as the distance between two points in one dataset is generally several meters, and that the coordinates are in the same reference system and therefore already very closely aligned. Since the ICP converges to the closest local minimum, it is very certain to produce correct results.

The more difficult case, that requires non-rigid matching, occurs when there was a loss of GPS signal while acquiring the LiDAR scans. For this case, we decided to adopt the GLMDTPS algorithm to our problem setting due to its impressive performance compared to other state-of-

the-art non-rigid point set registration algorithms and the high tolerance against outliers it showed in the tests presented in the paper by Yang et al. [14].

### 4.3.2. Implementation and Testing

As there are many open-source implementations for ICP, we decided to run two implementations on our data, to compare the results. For one, we picked the ICP implementation provided in CloudCompare, and as the second, the point-to-point version included in the Open3D package for Python.

For first testing, we manually identified the correct matches between the already classified, non-distorted “Werk2\_classified\_part1” file and the first GCP file. From 30 pole ground points extracted from the “Werk2\_classified\_part1” file, seven have matches among the GCPs. This is the expected result.

Both implementations of ICP resulted in the same seven matches and a transformation very close to the identity transformation. For further comparison, we also used a simple nearest-neighbor matching on the same datasets, which produced the exact same matches, hence rendering the ICP unnecessary except for the slight rigid transformation.

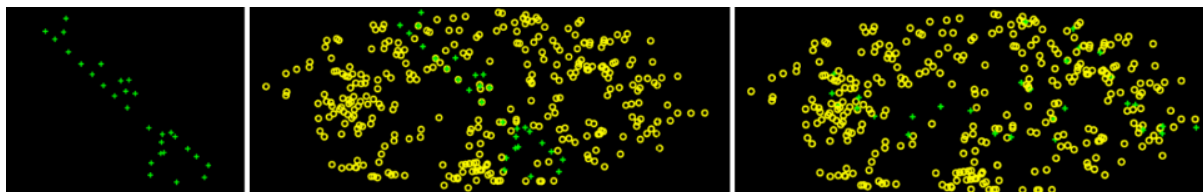


Figure 17: Demonstration of GLMDTPS. Left: PCPs from “Werk2\_classified\_part1”. Middle: PCPs and local GCPs from “GCPs set 1” before GLMDTPS. Right: After GLMDTPS.

For GLMDTPS, there exists a demo implementation in MATLAB provided by Yang et al. [14]. Since it was not designed for use cases to match two separate datasets, it only takes one dataset as input, deforms it, and finds a transformation back to the original. We adapted this implementation to identify matches and transformation for two independent datasets.

Since all datasets provided to us are assumed accurate except for slight noise and we did not have distorted datasets to test with, we used the same data as for the ICP to gain a first impression of its performance. It was immediately clear that it does not provide correct results for our datasets due to the high ratio of points that do not have matches in both datasets. Furthermore, the local distance metric used is based on neighboring relations between points in one dataset. While this does not pose a problem even with many outliers in the target set in case that every point in the source set has a match in the target set, the amount of PCPs that do not have matching GCPs resulted in loss even of the matches that are correct and already very close to each other, as can be seen in Figure 17. On the left, there are the PCPs extracted from “Werk2\_classified\_part1”. The second picture shows the PCPs together with a local subset of the GCP set before the transformation, while the last picture shows the same sets after transformation. Although in the middle picture we can clearly see some correct matches, the right picture shows none.

However, given sets of data that only contain the matches found via nearest neighbor matching, the GLMDTPS came to the correct conclusions and the thin plate spline-based transformation resulted in a more exact correspondence between the two sets than the ICP could provide. Since



the GLMDTPS can also compute transformations for non-rigid cases, we decided to replace the ICP with nearest neighbor matching and a GLMDTPS transformation in all cases, despite the higher computational complexity.

Therefore, we decided to implement and test an iterative algorithm based on nearest neighbor matching and GLMDTPS in order to find the correct matches and compute a transformation that can be used to correct all points in the local LiDAR point cloud. The algorithm iterates between computing matches via a nearest neighbor, computing a transformation based on these matches with the GLMDTPS, and using this transformation on all PCPs. Then we repeat these steps with the transformed PCPs and the GCP set until no new matches are found or the resulting transformation delivers worse results.

**Table 4: Results of iterated GLMDTPS with different thresholds for nearest neighbor matching**

<i>Matching Threshold in cm</i>	<i>Matches</i>	<i>Distance after transformation</i>
100	Correct	Increased to avg 56 cm
90	Correct	Decreased to avg 31 cm
80	Incorrect	Decreased to avg 31 cm
		Original distance: avg 41 cm

Trials have shown that for the nearest neighbor-based matching, a matching threshold of 90 cm for the initial matching and 70 cm in each further iteration provide the best results for matching and transformation, as shown in Table 4. These results are based on the dataset “086b” as this matching proved to be the most difficult. For “Werk2\_classified\_part1”, all of the matching thresholds in the table provided the same results.

### 4.3.3. Results

This iterated GLMDTPS algorithm resulted in the correct matches for both the “Werk2\_classified\_part1” and “086b” datasets in combination with both GCP sets and provided a transformation that significantly lowered the average distance between matches, with one set of PCPs achieving considerably better results than the other with both sets of GCPs. Examples for the distances of matches before and after running the iterated GLMDTPS can be seen in Table 5, which describes the difference before and after the algorithm for two test cases. The left shows the matching of PCPs from “werk2\_classified\_part1” to “GCP set 1”, the right the matching of PCPs from “086b” to “GCP set 1”. On both sides, the first column contains the ID of the PCP, the second column the original distance to its matching GCP, and the third column the distance between the match after the transformation. The higher distance after the transformation in case of the “086b” file can be explained by the higher complexity of the set. Results from the matching to “GCP set 2” can be found in Appendix D, Table 1.

**Table 5: Distances of matches before and after iterated GLMDTPS. ID: ID of PCP. OD: Distance between PCP and matching GCP before transformation. ND: Distance after transformation.**

<i>ID</i>	<i>OD</i>	<i>ND</i>	<i>ID</i>	<i>OD</i>	<i>ND</i>
5	0.551	0.311	1	0.587	0.264
6	0.565	0.152	16	> 0.9	0.544
7	0.744	0.0516	17	0.519	0.168
9	0.741	0.231	40	0.473	0.105
10	0.421	0.045	47	0.270	0.503
24	0.613	0.163	49	0.351	0.306
27	0.513	0.155	55	0.199	0.243
			56	0.477	0.372
<i>avg</i>	<b>0.592</b>	<b>0.158</b>	<i>avg</i>	<b>0.411</b>	<b>0.313</b>

Since we did not have distorted datasets to test on, we artificially distorted the PCPs gained from “Werk2\_classified\_part1” with two affine transformations and sent the resulting datasets to our mentors at Airbus, who confirmed that these artificial distortions were useful examples for testing purposes. The distortion resulted in an average distance of 5.7 meters, respectively 8.4 meters, from the original point to the distorted point, within both cases four points within the 90 cm initial matching distance from their original. Our algorithm was able to completely reconstruct the original coordinates from the artificially distorted ones. The results for these four points for one of the two datasets, including initial distance and distance after the algorithm, are shown in Table 6. A full table with coordinates and all distances from the same dataset can be found in Appendix D, Table 2.

**Table 6: Distances between matches of artificially distorted data before iterated GLMDTPS and after. ID: ID of distorted PCP. OD: Distance between distorted point and original point before transformation. ND: Distance after transformation.**

<i>ID</i>	<i>OD</i>	<i>ND</i>
0	0.015	0
1	0.072	0
2	0.784	0
24	0.788	0

Using the transformation gained from the algorithm on the distorted points that were not within matching distance for the initial matching also correctly reconstructed them and matched them to their undistorted counterpart.

A weakness that we have not yet addressed in our implementation is that our algorithm assumes that, even with distorted datasets, there exist initial matches to base a first transformation on, which might not be true in case there was an extremely distorted GPS signal in course of the whole scanning process of the street. However, this might be fixed if the user can manually provide a few initial matches for a first transformation.

With these results, we gained an algorithm that provides a transformation which can be used to attempt geometric correction of all points in the LiDAR point cloud.

#### 4.4. Software Prototype

In order to integrate all the algorithms together and have the pipeline working from raw data to the creation of the HD map, we built a small software in Python.

The first function of the software is to compute the segmentation and the classification of a point cloud and to display it. It then extracts the PCPs and displays them in another viewport. Furthermore, the matching process will be run in the background and the correspondence between PCPs and GCPs will be shown in the third viewport. Finally, the deformation of the input point cloud is computed in regard to the best matching between PCPs and GCPs with minimum distance.

As the segmentation and classification part takes a long time to compute, the software is designed in two modes: demo and test. The demo mode plots the four steps for the dataset that we worked with. Every output was computed by us previously before saved, so the software just needs to load the results to present a fast and functional demo.

Nevertheless, it is possible to test the full pipeline to create the HD map for new input data using the test mode. The user can choose either the RandomForest model or PointNet++ architecture we trained before to segment and classify their own datasets of the point cloud.

We noticed that the visualisation of the point clouds with the matplotlib library is very time-consuming to display a big scene and the plotted points are hard to distinguish due to poor resolution. Therefore, we added an option to visualize the results with CloudCompare, an open-source software.

See Appendix E for the full documentation and screenshots of the software.

## 5. Conclusion

In this report, we have presented two different frameworks for point cloud classification. They are divided into classic machine learning, which is based on point sampling, neighborhood recovery, and features extraction, with which then classic machine learning algorithms such as Random Forest are applied, and a deep learning approach, which uses PointNet++. Using the two different frameworks, we have analyzed the main differences regarding applying both methods for LiDAR dataset classification. The classic machine learning framework provides the more robust and computationally efficient approach without the need for GPU access for computation. On the other side, the deep learning approach is more time-efficient and designed for real-time classification. Nevertheless, further improvement for both algorithms is to train them on different types of data, so they become robust to any kind of area.

In order to make use of the segmentation for matching with GCPs, we have implemented a method to extract the PCPs from the pole-like structures that were produced in the previous step. For this, we have chosen to use DBScan instead of K-means because it can easily be applied to sceneries with promising results without knowing the number of poles beforehand.

For the matching between the Pole Control Points extracted from the mobile data and the Ground Control Points gained from the radar satellite data, we have presented an algorithm based on different point set registration methods that correctly identifies matches between the two sets and produces a thin-plate-spline based transformation that can be used to correct the coordinates of all points in the point cloud from the LiDAR dataset.

An impulse for further optimization is to investigate the potential of replacing the GLMDTPS with another algorithm that provides a TPS based transformation based on two already matched sets, to get a more computationally efficient result, since we are already aware of the correct matches when the GLMDTPS is called. Furthermore, it might be possible to get an even more accurate transformation with further tuning of the transformation specifically for this matching problem. We have summarized ideas of improvement in Appendix F, Table 1.

In conclusion, in the course of this project we have developed an algorithm that, provided with an unclassified LiDAR point cloud and a set of GCPs, results in a geometrically corrected version of the point cloud, and while the result is certainly not perfect yet, it is a very good base for further research into this direction.

## Bibliography

- [1] S. Montazeri, C. Gisinger, M. Eineder and X. X. Zhu, "Automatic Detection and Positioning of Ground Control Points Using TerraSAR-X Multiaspect Acquisitions," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 5, pp. 2613-2632, 15 2018.
- [2] E. Grilli, F. Menna and F. Remondino, "A review of point clouds segmentation and classification algorithms," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 2017.
- [3] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.
- [4] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer and S. Silvio, *3d semantic parsing of large-scale indoor spaces*, Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2016.
- [5] Y. Wang, T. Shi, P. Yun, L. Tai and M. Liu, "PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud," 17 7 2018.
- [6] C. R. Qi, L. Yi, H. Su and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," 7 6 2017.
- [7] G. Elbaz, T. Avraham and A. Fischer, "3D point cloud registration for localization using a deep neural network auto-encoder," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.
- [8] J. Huang and S. You, "Point cloud matching based on 3D self-similarity," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012.
- [9] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239-256, 1992.
- [10] S. Gold, R. A. Rangarajan, C.-P. Lu, S. Pappus and E. Mjolsness, "NEW ALGORITHMS FOR 2D AND 3D POINT MATCHING: POSE ESTIMATION AND CORRESPONDENCE Point-matching Pose estimation Correspondence Neural networks Optimization Softassign Deterministic annealing Affine transformation," 1998.
- [11] G. P. Penney, P. J. Edwards, A. P. King, J. M. Blackall, P. G. Batchelor and D. J. Hawkes, "A stochastic iterative closest point algorithm (StochastICP)," in *Lecture*

*Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001.

- [12] J. Yang, H. Li, D. Campbell and Y. Jia, "Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2241-2254, 11 2016.
- [13] H. Chui and A. Rangarajan, "A new point matching algorithm for non-rigid registration," *Computer Vision and Image Understanding*, vol. 89, no. 2-3, pp. 114-141, 2003.
- [14] Y. Yang, S. H. Ong and K. W. C. Foong, "A robust global and local mixture distance based non-rigid point set registration," *Pattern Recognition*, vol. 48, no. 1, pp. 156-173, 11 2015.
- [15] M. Weinmann, B. Jutzi, C. Mallet and M. Weinmann, "GEOMETRIC FEATURES and THEIR RELEVANCE for 3D POINT CLOUD CLASSIFICATION," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017.
- [16] F. A. Limberger and M. M. Oliveira, "Real-time detection of planar regions in unorganized point clouds," *Pattern Recognition*, vol. 48, no. 6, pp. 2043-2053, 11 2015.
- [17] M. Weinmann, B. Jutzi and C. Mallet, "Semantic 3D scene interpretation: A framework combining optimal neighborhood size selection with relevant features," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vols. II-3, pp. 181-188, 7 2014.
- [18] A. Myronenko, X. Song, M. A. Carreira-Perpiñán and P. Perpiñán, "Non-rigid point set registration: Coherent Point Drift".
- [19] T. Hackel, J. D. Wegner and K. Schindler, "FAST SEMANTIC SEGMENTATION of 3D POINT CLOUDS with STRONGLY VARYING DENSITY," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2016.
- [20] J. Balado, L. Díaz-Vilariño, P. Arias and L. M. González-Desantos, "Automatic LOD0 classification of airborne LiDAR data in urban and non-urban areas," *European Journal of Remote Sensing*, vol. 51, no. 1, pp. 978-990, 11 2018.
- [21] M. Weinmann, M. Weinmann, C. Mallet and M. Brédif, "A classification-segmentation framework for the detection of individual trees in dense MMS point cloud data acquired in urban areas," *Remote Sensing*, vol. 9, no. 3, 11 2017.

## Appendix

### Appendix A: Gitlab Repository and Responsibilities

The complete implementation is to be found on the following Gitlab repository <https://gitlab.lrz.de/3d-matching-terrasar-dgcp/3D-Matching-Prj>.

The master branch contains some *README* files and some manuals while the code has been completely deployed on the master branch. In addition, the python development environment is defined under *environment.yml*, which can be used to create an identical development conda environment.

Research and implementation responsibilities:

Louise Loesch	Deep Learning segmentation, extraction of PCPs, software development
Islam Mansour	Point of contact, Milestones Scheduling, Project Plan, Machine Learning & segmentation, Data preprocessing for ML, Pipeline for feature creation, Linux Cluster, JupyterHub deployment, Infrastructure maintenance
Magdalena Reich	Matching and geometric correction from research to implementation
Xianbin Xie	Machine Learning segmentation, Extraction of PCPs, Data preprocessing (for ML, deep learning and Matching)

Writing responsibilities:

Abstract	Islam Mansour, Magdalena Reich
Introduction	Magdalena Reich, Xianbin Xie, Islam Mansour
Problem Definition and Project Plan	Islam Mansour
Datasets	
Point Cloud	Islam Mansour
Ground Control Points	Magdalena Reich
Segmentation of Mobile Mapping Data	Louise Loesch, Xianbin Xie
Matching of Mobile Data and Satellite-Derived Ground Control Points	Magdalena Reich
Literature Research	
Segmentation	Louise Loesch
Classification	Islam Mansour
3D Point Set Registration	Magdalena Reich
Approaches	
Segmentation and Classification	
Classical Machine Learning	Xianbin Xie, Islam Mansour
Deep Learning	Louise Loesch

Results	
Random Forest	Xianbin Xie
PointNet++	Louise Loesch
Discussion	
Extraction of light for matching	
Matching	Magdalena Reich
Selected Algorithms	
Implementation and Testing	
Results	
Software Prototype	Louise Loesch
Conclusion	Magdalena Reich, Louise Loesch, Xianbin Xie, Islam Mansour



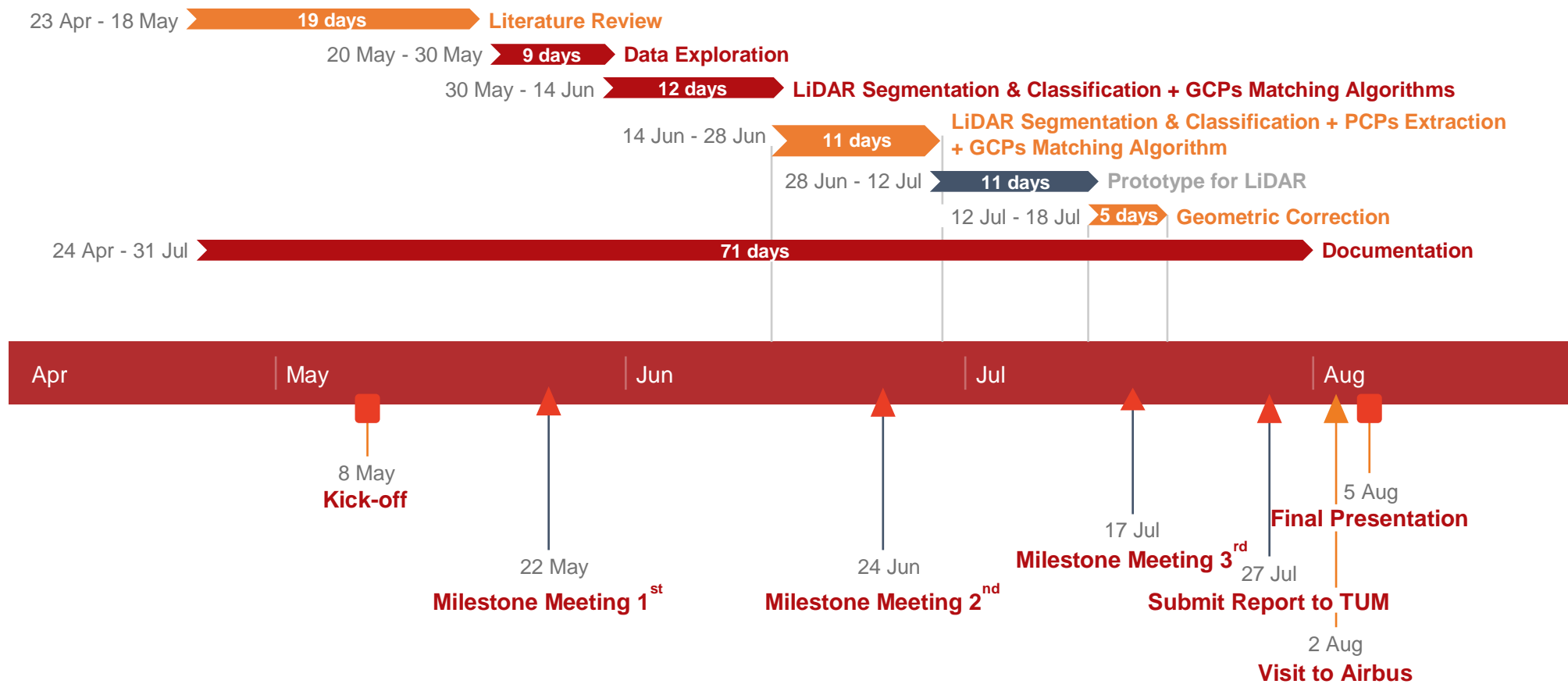


Figure 1: Project Plan Gantt Chart

## Appendix B: Comparison of Point Cloud file format

Table 2: Point Clouds File Formats <sup>1</sup>

TYPE	EXTENSION(S)	DESCRIPTION	READ	WRITE	BINARY/ASCII	POINT CLOUD(S)	MESH(ES)	OTHER	FEATURES
<b>BIN</b>	.bin	CloudCompare own format	X	X	binary	>1	>1	>1	Normals, colors (RGB), scalar fields (>1), labels, viewports, display options, etc.
<b>ASCII</b>	.asc,.txt,.xyz,.neu,.pts	ASCII point cloud file (X,Y,Z,etc.)	X	X	ASCII	1	0	0	Normals, colors (RGB), scalar fields (all)
<b>LAS</b>	.las	ASPRS lidar point clouds	X	X	binary	1	0	0	Colors (RGB) and various scalar fields (see LAS 1.4 specifications)
<b>E57</b>	.e57	ASTM E57 file format	X	X	mixed	>1	0	Calibrated picture(s)	Normals, colors (RGB or I), scalar field (intensity)
<b>PCD</b>	.pcd	Point Cloud Library format	X	X	binary	>1	0	0	Colors (RGB), normals, scalar fields (>1)
<b>PLY</b>	.ply	Stanford 3D geometry format (cloud or mesh)	X	X	both	1	1	0	Normals, colors (RGB or I), one ore several scalar fields, a single texture
<b>OBJ</b>	.obj	Wavefront mesh	X	X	ASCII	1	>1	Polyline(s)	Normals, materials and textures
<b>VTK</b>	.vtk	VTK file format (triangular mesh or cloud only)	X	X	ASCII	1	1	0	Normals, colors (RGB), scalar field(s) (>1)
<b>STL</b>	.stl	STereoLithography file format(mesh)	X	X	ASCII	0	1	0	Normals
<b>OFF</b>	.off	Object File Format (mesh)	X	X	ASCII	0	1	0	0
<b>FBX</b>	.fbx	Autodesk (Filmbox) File Format	X	X	ASCII or BINARY	0	>1	0	Normals, colors (RGB), materials and textures
<b>DXF</b>	.dxf	Autocad DXF format	X	X	ASCII	>1	>1	polyline(s)	Normals, colors (RGB)
<b>SHP</b>	.shp	ESRI Shape file format	X	X	binary	>1	0	Polyline(s), polygon(s), contour plot(s), etc.	Scalar fields (1 per entity)

<sup>1</sup> FILE I/O - CloudCompareWiki", Cloudcompare.org, 2019. [Online]. Available: [https://www.cloudcompare.org/doc/wiki/index.php?title=FILE\\_I/O](https://www.cloudcompare.org/doc/wiki/index.php?title=FILE_I/O). [Accessed: 27- Jul- 2019]

## Appendix C: Point Clouds Features Definition

$$\text{Linearity } L_\lambda = \frac{e_1 - e_2}{e_1}$$

$$\text{Planarity } P_\lambda = \frac{e_2 - e_3}{e_1}$$

$$\text{Sphericity } S_\lambda = \frac{e_3}{e_1}$$

$$\text{Omnivariance } O_\lambda = \sqrt[3]{e_1 e_2 e_3}$$

$$\text{Anisotropy } A_\lambda = \frac{e_1 - e_3}{e_1}$$

$$\text{Eigenentropy } E_\lambda = -\sum_{i=1}^3 e_i \ln(e_i)$$

$$\text{Sum } \Sigma_\lambda = e_1 + e_2 + e_3$$

$$\text{Surface Variation } C_\lambda = \frac{e_3}{e_1 + e_2 + e_3}$$

$$\text{Vertical range } z_{max} - z_{min}$$

$$\text{Height below } z - z_{min}$$

$$\text{Height above } z_{max} - z$$

## Appendix D: Additional Data from Matching

Appendix D, table 1: Matching results from “Werk2\_distorted\_part1” (left) and “086b” (right) to “GCP set 2”

Werk2_distorted_part1			086b		
ID	Old dist	New dist	ID	Old dist	New dist
3	0.369245	0.059914	12	0.719338	0.269985
4	0.33859	0.264001	16	> 0.9	0.290413
5	0.500798	0.123644	47	0.477042	0.157817
6	0.612463	0.246414	49	0.427185	0.167782
7	0.80341	0.078584	58	0.45697	0.15696
9	0.658169	0.110414	68	0.327925	0.158512
27	0.589117	0.13996	74	> 0.9	0.626156
<b>Avg dist</b>	<b>0.553113</b>	<b>0.146133</b>	<b>Avg dist</b>	<b>0.481692</b>	<b>0.261089</b>

Appendix D, table 2: Artificially distorted data before and after matching

ID	Coordinates before transf.				Coordinates after transf.				Match
	X	Y	Z	Dist	X	Y	Z	Dist	
0	535016.477	5278932.456	455.984	0.015	535016.477	5278932.456	455.984	0.0	0
1	535012.878	5278930.911	457.949	0.072	535012.878	5278930.911	457.949	0.0	1
2	535053.412	5278904.430	455.682	0.785	535053.413	5278904.430	455.682	0.0	2
3	535080.684	5278915.946	455.324	1.293	535080.684	5278915.946	455.324	0.0	3
4	535122.614	5278933.093	455.192	2.123	535122.614	5278933.093	455.192	0.0	4
5	535162.148	5278949.454	455.371	2.919	535162.148	5278949.454	455.371	0.0	5
6	535197.013	5278963.638	455.358	3.626	535197.013	5278963.638	455.358	0.0	6
7	535232.101	5278977.952	455.590	4.338	535232.101	5278977.952	455.590	0.0	7
8	535265.856	5278991.608	455.515	5.024	535265.856	5278991.608	455.515	0.0	8
9	535305.835	5279007.793	455.516	5.838	535305.835	5279007.793	455.516	0.0	9
10	535372.042	5279034.796	455.334	7.187	535372.042	5279034.796	455.334	0.0	10
11	535392.732	5279052.366	458.165	7.622	535392.732	5279052.366	458.165	0.0	11
12	535418.407	5279051.466	455.744	8.129	535418.407	5279051.466	455.744	0.0	12
13	535453.474	5279054.303	459.307	8.122	535453.474	5279054.303	459.307	0.0	13
14	535456.583	5279050.631	456.114	8.883	535456.583	5279050.631	456.114	0.0	14
15	535488.158	5279038.627	456.600	9.495	535488.158	5279038.627	456.600	0.0	15
16	535527.966	5279037.015	457.272	10.285	535527.967	5279037.015	457.272	0.0	16
17	535550.071	5279070.050	457.568	10.762	535550.071	5279070.050	457.568	0.0	17
18	535533.836	5279105.652	457.592	10.494	535533.836	5279105.652	457.592	0.0	18
19	535494.853	5279110.519	457.594	9.735	535494.853	5279110.519	457.594	0.0	19
20	535467.716	5279095.905	457.150	9.174	535467.716	5279095.905	457.150	0.0	20
21	535439.135	5279084.118	456.922	8.591	535439.135	5279084.118	456.922	0.0	21
22	535401.815	5279068.766	457.507	7.829	535401.815	5279068.766	457.507	0.0	22
23	535388.195	5279063.077	457.352	7.551	535388.195	5279063.077	457.352	0.0	23
24	535055.841	5278927.509	455.452	0.788	535055.841	5278927.509	455.452	0.0	24
25	535215.823	5279009.808	460.885	4.064	535215.823	5279009.808	460.885	0.0	25
26	535161.668	5278974.779	455.692	2.937	535161.668	5278974.779	455.692	0.0	26
27	535258.400	5279017.648	455.922	4.916	535258.400	5279017.648	455.922	0.0	27
28	535252.529	5279005.835	458.099	4.780	535252.529	5279005.835	458.099	0.0	28
29	535217.006	5279000.096	458.692	4.070	535217.006	5279000.096	458.692	0.0	29

## Appendix E: Software Documentation

### Purpose of the software

This software regroups all the different algorithms we developed and tried in the context of the Airbus project. This software includes :

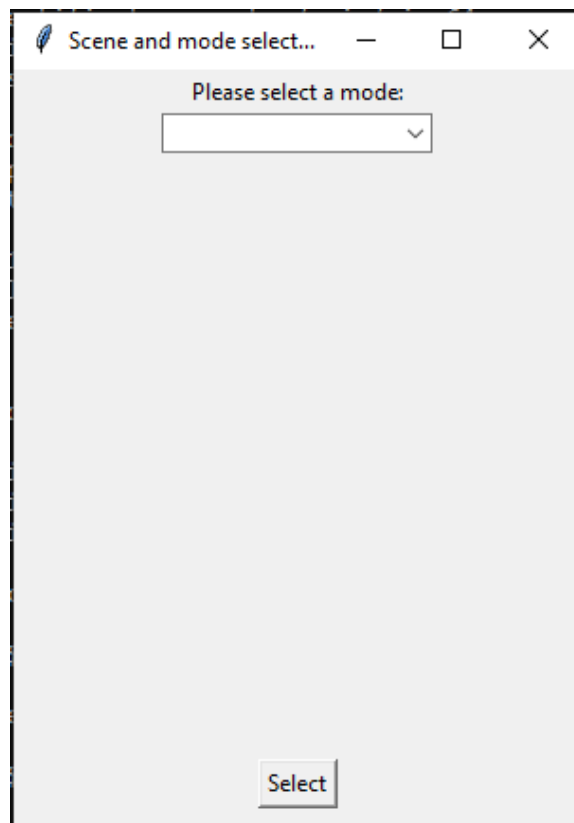
- A demonstration part, where the segmentation of the point cloud, the extracted light points, the matching with GCP and the deformation with GCP have been precalculated and are displayed as such.
- A test part, where the same results are displayed after their computation.

### Overall presentation

The software is decomposed in three parts: the home page, which one access when launching the software, the demonstration part and the test page.

#### Home page

When launching the software, one should encounter the following screen:

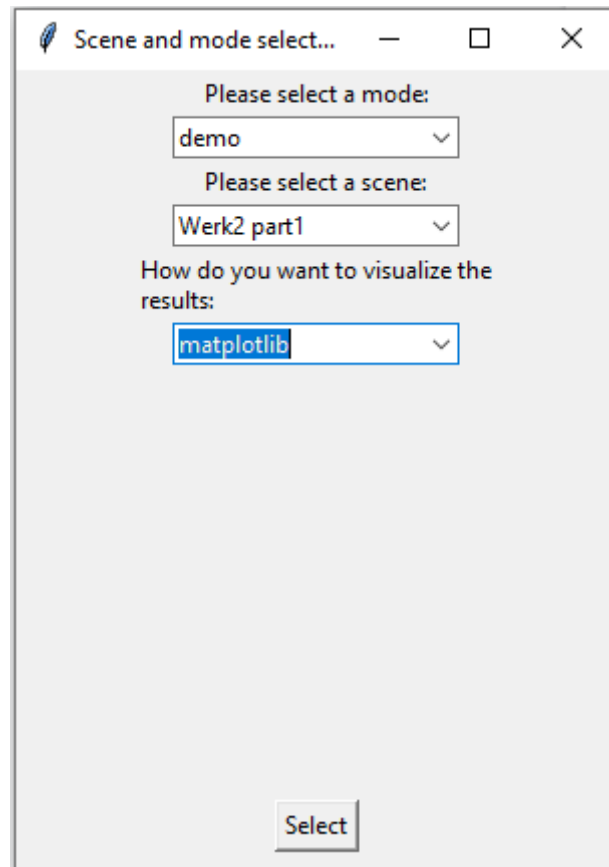


Home page: default screen

Starting from this point, one should make the choice of going for the demonstration mode, or for the test mode.

## Demonstration mode

When going for the demonstration mode, one has two choices to make: the selection of the precomputed scene, and the tool used to visualize the results.



The image shows a dialog box titled "Scene and mode select...". It contains three dropdown menus and a "Select" button. The first dropdown menu is labeled "Please select a mode:" and has "demo" selected. The second dropdown menu is labeled "Please select a scene:" and has "Werk2 part1" selected. The third dropdown menu is labeled "How do you want to visualize the results:" and has "matplotlib" selected. The "Select" button is located at the bottom center of the dialog box.

Home page: demo mode

## Selection of the scene

When arriving to the home page and when one has selected the demonstration mode, one has to choose a scene. You have the choice between three different german streets:

- 086b
- Werk2 part 1
- Werk2 part 2

When choosing the scene, the following choice to make appears.

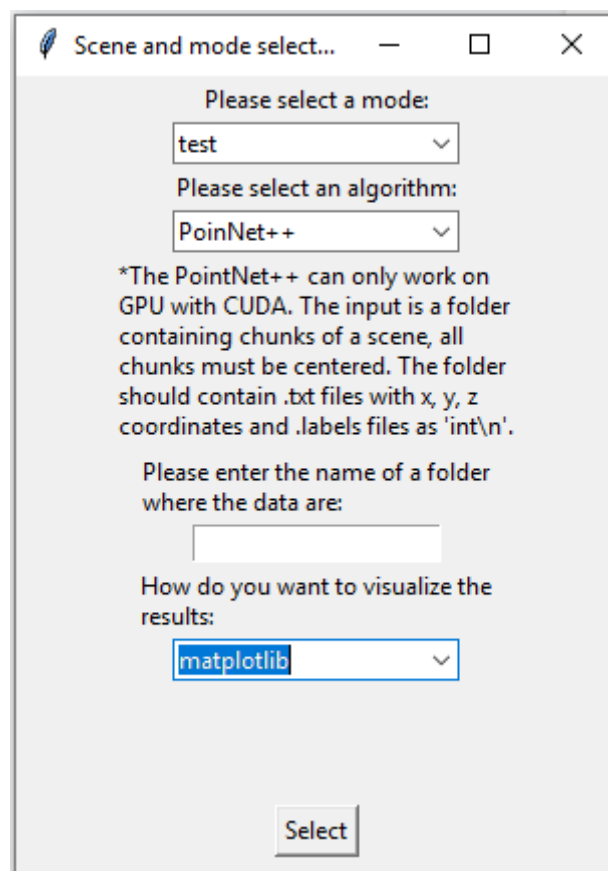
## Selection of the visualization tool

When arriving at the home page and when one have selected the demonstration mode and the scene, one has to choose a visualization tool. You have a choice between two visualization tools:

- matplotlib - which will then be included directly into the software, but will be really slow to load.
- [CloudCompare](#) - which is free software that loads the points faster and has a more logical presentation of the points.

## Test mode

When going for the test mode, one has two choices to make and one file to specify: the selection of the segmentation algorithm to use, of the visualization tool and specification of the location of files needed by the algorithm.



The screenshot shows a dialog box titled "Scene and mode select...". It contains the following elements:

- A label "Please select a mode:" followed by a dropdown menu with "test" selected.
- A label "Please select an algorithm:" followed by a dropdown menu with "PoinNet++" selected.
- A paragraph of text: "\*The PointNet++ can only work on GPU with CUDA. The input is a folder containing chunks of a scene, all chunks must be centered. The folder should contain .txt files with x, y, z coordinates and .labels files as 'int\n'".
- A label "Please enter the name of a folder where the data are:" followed by an empty text input field.
- A label "How do you want to visualize the results:" followed by a dropdown menu with "matplotlib" selected.
- A "Select" button at the bottom.

Home page: test mode

## Selection of the segmentation algorithm

When arriving at the home page and when one has selected the test mode, one has to choose a segmentation algorithm to use. You have a choice between two algorithms:

- [Random Forest](#) - which is a Machine Learning algorithm that we pre-trained.
- [PointNet++](#) - which is a Deep Learning algorithm that we also pre-trained.

Please note that using the Deep Learning algorithm will require a functional GPU along with CUDA.

## Files needed by the algorithm

Both algorithms will require you to specify additional files.

If you choose to use **PointNet++**, it will be asked for you to specify the folder in which the different data chunks are stored.

Please note that those chunks have to be centred.

If you choose to use **Random Forest**, it will be asked for you to specify the LAS file, containing the (x,y,z) coordinates of the 3D scene.

## Selection of the visualization tool

When arriving at the home page and when one have selected the test mode and the scene, one has to choose a visualization tool. You have a choice between two visualization tools:

- matplotlib - which will then be included directly into the software, but will be really slow to load.
- [CloudCompare](#) - which is free software that loads the points faster and has a more logical presentation of the points.

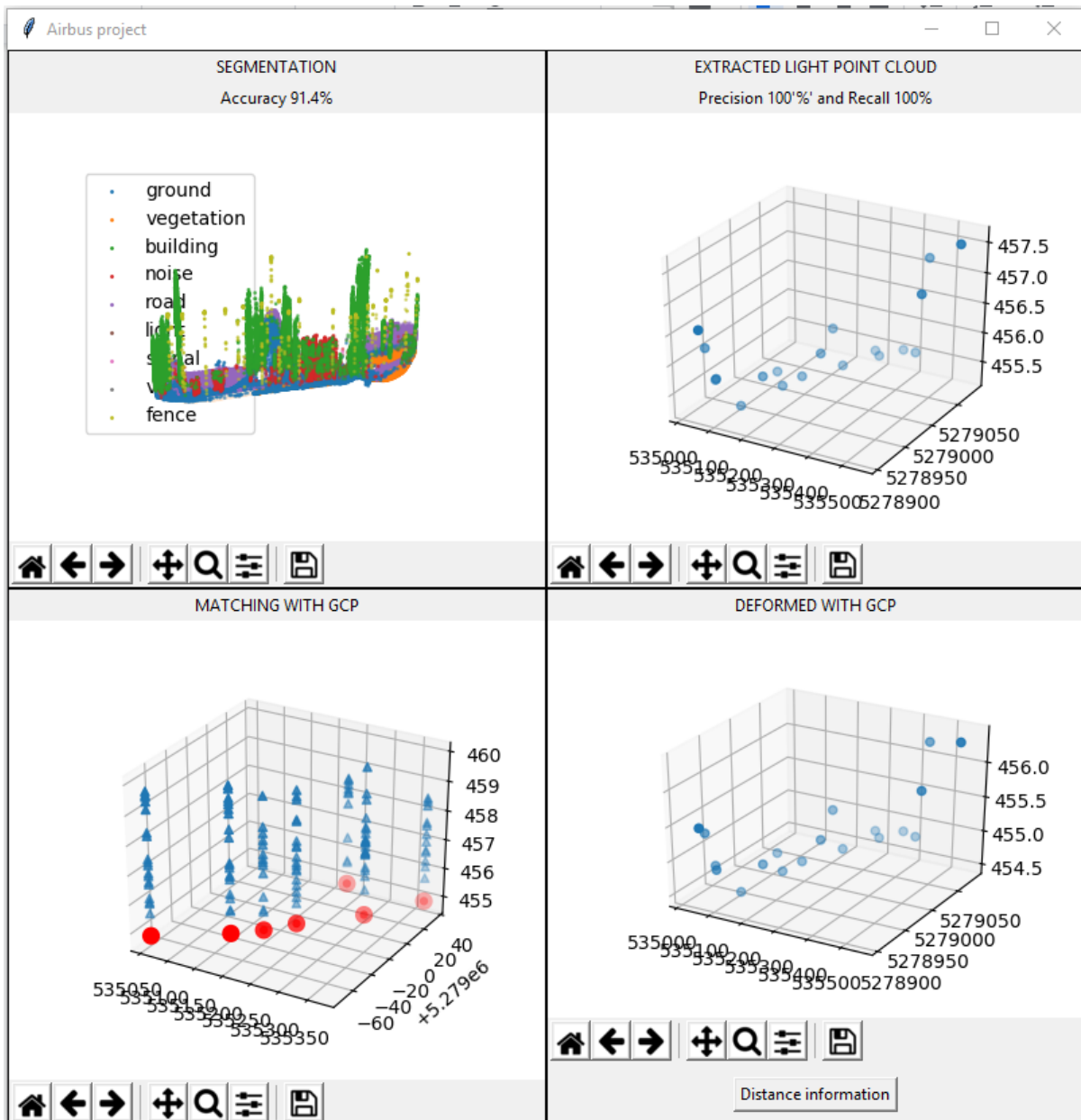
## Results page

When finishing the different choices you had to make in the home page, you will be redirected to the results page.

The result page is decomposed into 4 distinct parts:



- The first part is composed of the segmentation, where you'll see the accuracy in case you are in the demonstration mode, and the segmented point cloud, where the different classes are shown using different colours.
- The second part is composed of the extracted light point cloud. The precision and the recall will also be displayed in case you are in the demonstration mode.
- The third panel is matching with GCP.
- The fourth and last part is the deformed point cloud using GCP. In this part, you will also see a button named *Distance information* (cf Distance Information part).



Result page

When choosing **matplotlib** as a visualization tool, the different point clouds will be interactive, which means one can click on the figures to navigate the point cloud.

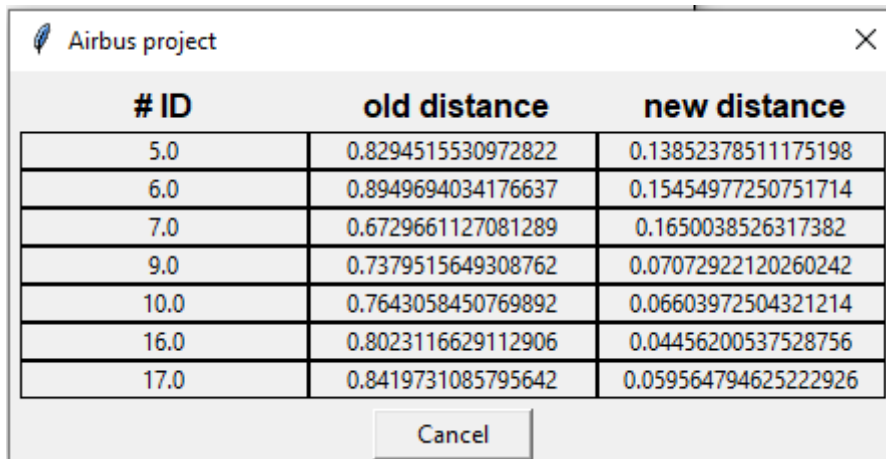
When choosing **CloudCompare** as a visualization tool, one will have to choose to display the x, y and z coordinates and will then be able to navigate the point cloud.

Please note that the accuracy, the precision and the recall won't be accessible in the test mode, as one doesn't necessarily have access to the ground truth.

### Distance information

In the fourth panel, that is "Deformed with GCP", you will notice a button named "Distance information".

Clicking on this button will make a new window appear, containing the different distances computed during the deformation. One will then be able to compare the old distance and the new distance.



# ID	old distance	new distance
5.0	0.8294515530972822	0.13852378511175198
6.0	0.8949694034176637	0.15454977250751714
7.0	0.6729661127081289	0.1650038526317382
9.0	0.7379515649308762	0.07072922120260242
10.0	0.7643058450769892	0.06603972504321214
16.0	0.8023116629112906	0.04456200537528756
17.0	0.8419731085795642	0.059564794625222926

Result page: Distance information

## Appendix F: Potential improvement of different methods

Table 1: Potential improvement of the different approaches

Random Forest and Pdal Pipeline	The overall pipeline of point cloud processing does not support multi-thread and parallelization. Therefore, the processing time can be decreased by supporting parallelization.
PointNet++	We could have used another approach with PointNet++: train on one scene, use a second for validation and the third for testing. It is important to normalize the coordinates for each scenes. With this kind of training, we can now test if the model is robust to the size of the cloud.
Extraction	-Improve the fine-tuning of the parameters (epsilon and min_samples) of DBSCAN for clustering. -Create or find a machine learning or deep learning approach for extraction of the light pole. Maybe include this architecture to the segmentation approach.
Matching	Replace GLMDTPS by an algorithm that only computes the TPS-based transformation based on given matches, tune the TPS for more accurate transformation, give option to manually provide first matches in case nearest neighbor does not find correct ones (strong distortion of whole dataset).