FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Immersive Redesign

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

BSc Medieninformatik und Visual Computing

eingereicht von

Ahmed El Agrod
Matrikelnummer 11811340

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass.(FWF) Dr. Stefan Ohrhallinger

Wien, TT.MM.JJJJ

(Unterschrift Verfasserin)

(Unterschrift Betreuerin)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Immersive Redesign

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

BSc Media Informatics and Visual Computing

by

Ahmed El Agrod
Registration Number 11811340

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass.(FWF) Dr. Stefan Ohrhallinger

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Ahmed El Agrod

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Danksagung

Hier fügen Sie optional eine Danksagung ein.

Acknowledgements

Optional acknowledgements may be inserted here.

Kurzfassung

Mit der Zeit steigt nicht nur die Rechenleistung von Rechnern sondern auch Scan Techniken werden besser. Dadurch wird es immer leichter eine Umgebung zu scannen und ein 3D Modell von dieser zu erstellen. Heutzutage sind schon Smartphones und Dronen in der Lage einen Bereich zu scannen. Hierbei ist zu beachten, dass die Ergebnisse mit hoher Wahrscheinlichkeit schlechter als die von bestimmten Scannern ist. Um den Leuten nun zu ermöglichen ihre Modelle kombinieren und bearbeiten zu können, haben mein Kollege Manuel Keilman und ich eine Software dafür geschrieben. Hierbei ist es wichtig, dass es sich bei den Modellen um Punktwolken handelt. Mit der Software können Punktwolken zusammengefügt und Bereiche extrahiert werden. Diese Bereiche können verschoben oder gelöscht werden. Unser primäres Ziel sind Punktwolken von Straßenszenen die mittels LiDAR gescannt wurden. Um zwischen den Objekten einer Straßeszene unterscheiden zu können, haben wir einen Klassifizierungsalgorithmus namens GRanD-Net [13] benutzt. Dieser clustered jedoch nicht die Punktwolke, weshalb ein weiterer Algorithmus hierfür notwendig ist. Um nun zwischen Objekten derselben Klasse unterscheiden zu können, benutzen wir DBScan [15]. Sowohl die Klassifizierung als auch das Clustern müssen nur einmal ausgeführt werden, bevor die Punktwolke hochgeladen wird. Der Vorteil beim Clustern von Punkten mit ähnlichen Eigenschaften ist, dass Nutzende nicht immer die Punkte selber auswählen müssen, womit die Usability verbessert wird. Die Ergebnisse vom Klassifizieren und Clustern tendieren leider dazu, nicht überall gute Ergebnisse zu liefern. Hierbei können Nutzende jedoch unsere Software benutzen, mit der sie Fehler selbst verbessern können. Unsere Software soll demnach Nutzenden ermöglichen ihre Punktwolken zu analysieren und sie leicht anzupassen.

Abstract

With scanning technologies and the processing power of computers increasing it is getting easier to scan the surrounding area and upload a 3D model of it. Nowadays mobile phones and drones are capable of doing that. However, the results most likely are not as good as dedicated scanners. To offer people an option to explore their scanned objects my colleague Manuel Keilman and I wrote a software with which users can upload multiple point clouds. Those point clouds can then be modified by adding, deleting, or moving parts of it. Our primary target is point clouds of street scenes. We selected an existing classification algorithm called GraND-Net [13] to identify objects of the scene. Since the classification algorithm does not cluster the classes a separate algorithm for that was needed. That is why we use DBScan [15] in order to distinguish different objects of the same class. Both steps are done before starting the application since they only have to be done once. The benefit of grouping points with similar features is that users do not have to manually select everything which improves the usability of our software. After classifying as well as clustering users can upload their point clouds and then select objects in order to modify them. Mistakes tend to happen which is why our software also offers users the option to manually classify whatever they want. All of that should make it easier for users to analyze and modify their point clouds.

Contents

CHAPTER 1

Introduction

Scanning 3D objects and loading them in a program nowadays is not as much of a hurdle as it was a couple of years ago. Computers have higher processing power and software is written more efficiently. The equipment needed to create 3D objects is not too expensive either. Certain mobile phones or drones are able to scan an area. One of the main ways to scan objects is to use LiDAR which utilizes light in order to scan an area. This thesis mainly focuses on 3D point clouds of real-life street scenes in Vienna which have been caught with LiDAR. Point clouds are 3D objects that only contain information about the coordinates and colors of points. [24]
The goal of this thesis, therefore, is to write a software with which users can modify those scanned point clouds. They should be able to move, delete and combine areas of a single or of multiple point clouds together. Scanning an environment tends to end up with holes in some areas. In case there are holes in the ground there is an option to fill those out. In order to do that the ground has to be detected first and that's where the RANSAC algorithm comes in handy. Filling out holes can then be done with an Image Inpainting algorithm. Identifying the ground is not only important in order to fill holes but also to place objects at the same height. Those algorithms and their use cases in our software were carried out by my colleague Manuel Keilman and are therefore discussed in his thesis in detail. [18]
In order to make it easier to modify point clouds, there is an option to classify them. With classifying and then subsequently clustering a point cloud, points would be assigned to different clusters which are also referred to as objects in this thesis. Objects, in this case, are cars, trees, buildings, humans, the ground, and poles. Being able to choose objects makes it much easier to apply one of the modifying operations that were mentioned. Users do not have to create a box and fit it to select certain points to change the scene. Classifying can be done automatically with the classification algorithm GRanD-Net [13] which will be explained further in the upcoming chapters. If one wants to use that option they would have to run GRanD-Net on their point cloud as a preprocessing step.
After classifying the cloud, it is clustered so that it is easier to distinguish and select objects of the cloud. For clustering, the DBScan [15] algorithm was used as it does not need any information about the actual numbers of clusters. Mistakes tend to happen when clustering as well as when

classifying but that's where the software should come in handy to make fixing those mistakes easier. In case the results are not satisfying users can still classify and cluster areas manually by creating a box that can be modified. All the points within that box are selected. Once points are selected they can be classified and changed like objects.
Immersive Redesign could open up a lot of opportunities to create a scene and experiment with it. Being able to modify a scene enables users to analyze it easier. Users could upload a scan of an empty street with its surrounding buildings and vegetation. If they want to know what it would be like if vehicles were on there they could simply add clouds of some vehicles and place them where they want.
As of now the classification and clustering results tend to have mistakes where areas are not classified properly or clustered optimally. However, when that is not a problem anymore users could do the preprocessing steps on a huge scene and then upload it to the software. Then they have an overview of where all the vehicles or trees are which should end up saving some time when analyzing a scene.
In the next section, the algorithms and certain terms will be explained further. After that, the methodology of the parts I was responsible for will be discussed in detail. Following that certain aspects such as run-time and visual appearance of the mentioned algorithms and our software will be evaluated. Lastly, options that could improve the current version of our software will be mentioned with a brief explanation of why they could be helpful.

CHAPTER 2

Related Work

2.1 Point Clouds

Point clouds are objects which consist of unconnected points in the 3D space. Due to point clouds only containing points and no other information it is relatively easy to create those. One simply needs a sensor such as LiDAR with which an environment can be scanned. Afterwards, the collected information can be stored in a laz file and then be used in our software to view it for example. [24]

2.2 GRanD-Net Algorithm

GRanD-Net is a semantic segmentation network for point clouds with millions of points. The algorithm only requires a point cloud as an input. After uploading the point cloud it is reduced to a constant size by performing grid-subsampling. In order to assign a label to every point and not only the subsampled ones, the indices will be tracked for up-sampling. Afterwards, the data is uploaded in multiple batches. To improve the learning process the data is given randomly rather than in order. Then the k-Nearest Neighbours (kNN) algorithm is applied with k predefined points of all the subsampled points. In case the subsampled points are fewer than the k neighbours the k points are substituted with the subsampled points. In order to assemble a batch of point clouds, indices for all the adjacent points are created. With those indices, relative features can be extracted. After that, dilated convolution and Gaussian error linear unit (GeLu) activation functions are used to create the model.
Convolution is used to extract low-level features in the first layer as well as higher-level features in the network. Dilated convolution adds a parameter with which the hole in a kernel can be modified which improves performance since the receptive area of a kernel is enlarged. In GRanD-Net dilated convolution is used to process irrelevant data faster while maintaining the necessary features.
GeLu considers the sign and the magnitude of an input. The input is multiplied with a value

which is evaluated stochastically depending on the input. GeLu is used on the outputs of the neural network. The benefit of using GeLu is that non-linear features can be learned better.
Afterwards, the dilated residual blocks are stacked. The result of that is up-sampled and run through multi-layer perceptrons as well as fully connected layers. That method makes sure that the labeling is not faulty. During up-sampling, the labels are processed step by step contrary to a simple interpolation which does not consider the distinction of classes.
After up-sampling conditional random field (CRF) is applied. CRF processes labels based on the coordinates of the input point and the labels of adjacent points. Furthermore, a cost function is used to improve the accuracy. [13]

Alternative Algorithms

Below a couple of alternative algorithms that were used on the SHREC Dataset will be mentioned. According to the paper [20], those did not perform better regarding overall classification accuracy as well as classification accuracy for each of the classes. A table containing the accuracy values is displayed in Figure 3.1. That ended up being the reason for choosing GRanD-Net over any of those.

PointNet++

PointNet++ is an upgrade of the PointNet algorithm [22] that tries to learn deep features effectively by processing point coordinates. [20] That is done by using an ordered neural network that applies the PointNet algorithm recursively on a subsample of the point coordinates. Metric space distances are utilized in order to learn local features. [23] PointNet++ was introduced in 2017 whereas GRanD-Net was published in 2020. That is likely the reason for PointNet++ performing worse than GRanD-Net as GRanD-Net utilizes the given information better.

P4UCC

Progressive 4-staged Urban Cloud Classifier (P4UCC) is a non-learning algorithm that consists of a 4-stage pipeline that classifies a point cloud sequentially. Despite P4UCC being a non-learning algorithm it performed better than some of the learning algorithms. That could be a fact due to some classes not being represented enough in the training dataset. For example, poles only make up 0.47% of the training dataset. Nonetheless, P4UCC ended up being worse than GRanD-Net. [20]

Spherical DZNet

Spherical DZNet converts a 3D point cloud into a 2D image and then tries to classify it with a convolutional neural network. The results are converted back into 3D space and interpolated for initial input. That method does seem to struggle with spherical projections. The object size changes depending on the distance to the sensors. That has a negative impact when training a model. Furthermore, that could the reason why that method struggled with classifying poles properly. [20]

ResGANet

ResGANet uses graphs to encode the spatial features of a point cloud. With that information, an end-to-end model is trained with a residual graph attentional network. That model predicts the labels of the point cloud. Similar to Spherical DZNet it struggled with classifying poles due to the small representation of poles in the training dataset. Besides, it performed worse than GRanD-Net. [20]

2.3 Laz Format

As mentioned previously LiDAR can be used to scan an environment. The collected data can then be stored in a laz file. The laz format is a lossless compression of the las format and usually is a tenth of the storage size of the las one. [5]
The las format consists of the following 3 parts: header, variable length records (VLR) and point records. The header stores information about the version and point format. The point format indicates which dimensions the file has. Dimensions are similar to attributes of a class. Examples would be 'x', 'y' and 'z' for the coordinates of the points. VLRs, however, contain information about the spatial reference system and dimensions that were added to the existing ones. Lastly, the point records store information about the point formats which define the existing dimensions of that format. [3]
Custom dimensions can be added from version 1.4 onwards but the values can only be integers or floats. That feature is beneficial since a dimension for cluster information can be created which does not exist otherwise. [4]

2.4 DBScan

The Density-Based Spatial Clustering of Applications with Noise algorithm (DBScan) is a clustering algorithm which identifies clusters and noise in spatial databases such as point clouds. The algorithm has 2 parameters 'eps' and the minimal amount of points needed for a cluster. The 'eps' value determines if two points are within the same neighbourhood. If the distance between both is greater than the 'eps' value they are not considered in the same neighbourhood and therefore not in the same cluster. Once a point has been assigned to a cluster it is not considered for other possible clusters anymore. However, if two clusters are determined to be close to each other they could be merged into one cluster. [15]

Choosing DBScan over k-Means

Another popular clustering algorithm is k-Means. In order to use k-Means the amount of desired clusters, x has to be given as an input. Afterwards random x points are chosen and considered as the center of a cluster. Then the distance for each point to the cluster centers is calculated. The points are assigned to the closest cluster. Once all points are assigned the center is recalculated. That process is repeated until the center of the clusters does not change. Even though k-Means

could be used as a clustering algorithm the fact that it needs the number of clusters as an input ended up being the reason it was not used since the number of clusters is unknown. [21]

2.5 Open3D

Open3D is an open-source library that is used for processing 3D data. It offers data structures such as points clouds, meshes, or RGB-D images and algorithms for C++ and Python. [25] In this thesis, a lot of functions for the point clouds were used. Furthermore, the GUI as well as rendering the scene and the data structures were created with that library. We chose this library since it is not difficult to use it and its functions are rather fast as they usually only take a couple of ms.
Some of the essential and frequently used functions for the point cloud operations are 'select_by_index', 'paint_uniform_color' and 'get_oriented_bounding_box'. The most important one being the 'get_oriented_bounding_box' which returns an 'oriented_bounding_box' object. [9] With the bounding box function 'get_point_indices_within_bounding_box' we are able to get certain indices of points of the point cloud. Once the indices of the points are known those points can be extracted with the already mentioned 'select_by_index' function. If one wants to transform the selected area they can call one of the given 'translate', 'rotate', or 'scale' functions. [10]
As mentioned the GUI was also created with that library. There are a couple of ways to implement a GUI with the library. We used the visualization.gui package since that one allowed us to easily customize our GUI. [11]

CHAPTER 3

Methodology

The purpose of this project is to first classify point clouds of real-life street scenes caught with LiDAR, then cluster them and subsequently be able to modify them in a program, which my colleague Manuel Keilman and I wrote. With our program, objects which are certain areas of a point cloud such as cars and buildings should be able to be moved and deleted. In order to move objects on the ground, a method for placing and translating them at the same height was necessary, which is why we use the RANSAC algorithm. Additionally, the option to fill holes in the ground by Image Inpainting is given. Since my colleague Manuel Keilman explains RANSAC and Image Inpainting as well as the reason for choosing those in his thesis [18], I will not go into further detail regarding those topics.
My part of the project was to find a classification algorithm specifically for street scenes which will be mentioned further in section 3.1. Since the algorithm we use does not automatically cluster the different classes a separate algorithm that does that was necessary. The reason why clustering is important is so that users can select single objects rather than having to select everything that is classified as a class but that will be explained further in the upcoming sections. Due to the fact that the number of actual clusters is unknown an algorithm that does not need that information was necessary. That is where DBScan [15] comes in handy which is a density-based clustering algorithm that only has the following inputs: the minimal number of points necessary for a cluster and a parameter 'eps' which is a number that specifies how close points should be next to each other to be considered of the same cluster. If the distance between two points is greater than the 'eps' value they will not be considered neighbors which means they do not belong together. Since the results were rather good and the run-time usually was within a couple of milliseconds we ended up using it. More regarding performance can be found in the 'Evaluation' chapter 4.2.
Lastly, my responsibility for the program was to figure out a way how users can load the classification and clustering results into the GUI which is done via laz files. Laz is a compressed type of the Las file format which is used to store LiDAR data. Along with its rather low memory consumption, the major benefit is that dimensions can be added, which has been done to store the clustering results. Dimensions are similar to attributes of a class that can be read once a laz

file has been loaded. Some pre-existing ones would be 'x', 'y', and 'z' for the coordinates or 'red', 'green', and 'blue' for the color information of the points.
Regarding the GUI, I also implemented that users can filter the cloud by the classes that were used in the SHREC dataset [1] which are 'building', 'car', 'ground', 'pole', and 'vegetation'. Since our dataset occasionally contains some human beings 'human' is also a filtering option. Furthermore, the option to set the class of marked areas in the point cloud is given. That is necessary due to misclassification by the classification algorithm and suboptimal clustering at some areas. Marking can be done by placing a box and then fitting it to contain all the desired points. All of those aspects will be discussed in detail in the upcoming sections.

3.1 Classification

For the classification, we used the GRanD-Net Algorithm [13]. On the Github page [19], the authors have a pretrained model based on the SHREC 2020 dataset [1]. This dataset contains 80 point clouds of which the ground truth is available. The choice for the algorithm was based on the fact that GRanD-Net performed better than any other algorithm on the mentioned dataset as seen in Figure 3.1. Since we use the given pretrained GRanD-Net model and the SHREC dataset contains 5 classes it is only possible to assign points to one of the 5 following classes: building, car, ground, pole, and vegetation.

Method	OA (%)	mIoU (%)	Building	Car	Ground	Pole	Vegetation
Baseline	91.30	66.39	82.52	40.13	89.10	39.46	80.72
P4UCC	94.13	72.25	84.35	60.51	96.46	40.18	79.75
Spherical DZNet	93.89	67.30	83.16	49.93	96.46	27.52	79.41
ResGANet	93.55	71.39	82.96	57.66	94.40	32.94	88.98
GRanD-Net	**97.83**	**86.40**	**93.66**	**83.92**	**98.10**	**61.79**	**94.55**

Figure 3.1: Algorithms used with the SHREC dataset [20]

In order to use the algorithm, the point cloud has to be represented in a txt file. The txt file is formatted as follows: each line represents a point and the coordinates are separated by whitespace. The first column represents the x-value, the second one the y-value, and the third one the z-value. The txt file can contain further columns such as color or label but those will not be taken into consideration as the algorithm only evaluates the first 3 columns. An example can be found in Figure 3.2.

```
87180.56967224 436592.25478027 2.49422762 3
87180.54867615 436592.24770020 2.49322745 3
87180.52766479 436592.23573730 2.49122760 3
87180.50666870 436592.23573730 2.49122760 3
87180.48567261 436592.22877930 2.49022767 3
87180.46467651 436592.23268555 2.49022767 3
87180.69366516 436601.03371094 2.70822761 3
87180.68467773 436601.03371094 2.70822761 3
```

Figure 3.2: Example txt format with x (1st column), y (2nd column) and z-values (3rd column), 4th column 'Label' will not be considered. Each line represents a point.

Since our dataset [6] consists of laz files, I wrote a script with which those can be converted into txt files is given (laz_to_txt.py). Additionally, the cloud can be subsampled via voxel subsampling which tends to improve the classification results for our dataset but for the SHREC dataset the results were worse. Even removing 0.3 % of a point cloud of the SHREC dataset ended up with a 40% worse result. Since there is no ground truth available for our dataset, the results were evaluated by the visual appearance. By subsampling with a factor of 0.5 some objects such as cars and buildings were classified more accurately which ends up in a better result. The average run-time for the classification algorithm took about 19 min and 10 sec and it did not matter how many points a cloud had. Since objects of our dataset were classified more accurately when subsampling it is beneficial to use it. A detailed review can be found in the 'Evaluation' chapter. When using voxel subsampling the cloud point coordinates are divided by 0.2 which equals the size of one grid cell. By dividing the coordinates by 0.2 each point can be assigned to one grid cell. After testing a couple of values the results for our dataset were the best for 0.2. Not every dataset has the same ratios of absolute values as our dataset which is why that value can be multiplied with a factor that users can type in to fit it for their dataset. If users want to apply a factor to the default number they simply have to type in the value for the input '–voxel_size'. The number will then be multiplied by 0.2. So if one types in 0.5 for 'voxel_size' the voxel size will be 0.2 * 0.5 which equals 0.1. The lower the voxel size the higher the number of points that remain. After calculating the voxel size the coordinates of the input point cloud are divided by the voxel size value and saved in an array 'grid_array'. In order to obtain the grid coordinate for each point, the calculated values in the 'grid_array' array are truncated by taking the value that comes after the decimal point. An example for the truncation part: if the grid coordinate was [2.5, 5.3, 0.7] then the value after truncating would be [2, 5, 0]. Afterwards all unique values of that calculation are evaluated. Lastly, each grid coordinate is saved as a key in a hashmap with its value being one of the points inside that grid.
After running the classification algorithm a file with the labels assigned to the points is created. The labels in this case are a number between 0 and 5 with the numbers representing a class. That file has as many lines as the input txt file but it only contains the label number. In order to know which classification belongs to which point the lines have to be matched. So the point in the txt file in the first line has the classification value of the first line in the labels file. In Figure 3.3 the numbers associated with the class can be seen with an example of a file created by the algorithm. That file can then be added onto a laz file by running the script add_label_to_laz.py.

Label: Class
0: undefined
1: building
2: car
3: ground
4: pole
5: vegetation

```
3       -> point in the 1st line is classified as ground
3
3
3
3
3
1       -> point in the 7th line is classified as building
1
1
5       -> point in the 10th line is classified as vegetation
5
5
```

Figure 3.3: Left: Label number with associated class. Right: Example of created file with labels and a brief explanation of how to interpret it

3.2 Clustering

After running the classification algorithm, the next step is to cluster the points of objects with identical classification id. In order to do that, DBScan is used, since the number of clusters is unknown. Instead of running DBScan on the whole cloud, it is divided into parts that only consist of one class. All points that are classified as 'car' for example end up as one point cloud and that process is repeated for each class. After splitting the cloud DBScan is applied to each class but the 'ground' class because the ground generally is one big piece and can therefore remain as a single object. Furthermore, users will not be moving or removing the ground as they would with cars for example. That method ends up with better clustering results than running DBScan once for the whole cloud but that will be discussed further in the 'Evaluation' chapter. In order to reduce run-time and improve clustering results, the split parts of the cloud that contain more than 105 000 points are subsampled via the same voxel subsampling as described in the classification section. 105 000 is the boundary since after testing some values that number ended up with the best results in terms of visual appearance and run-time. A good example would be that for a tree only one cluster is assigned as seen in Figure 3.5 rather than the top and bottom of that tree being different clusters as seen in Figure 3.4.

After running DBScan, usually, not all points are assigned to a cluster. However, since the number of unassigned points tends to be rather low and represents mostly very small sets of points that are not too close to actual objects, those are removed. An example can be seen in Figure 3.6. Before removing those points a bounding box for each cluster is created and all points inside are assigned the value of the cluster. That step is done because noise points are sometimes within the bounding box of an existing cluster. Instead of removing those, they are added to the existing cluster. In case two bounding boxes overlap the points are assigned to one of these. Basically, as soon as noise points are assigned to a cluster they will not be considered for other clusters. Even though minimizing the number of noise points could be achieved by changing the 'eps' value that will likely take more time since finding the proper value requires some testing. Furthermore changing the 'eps' value might improve one area of the cloud but

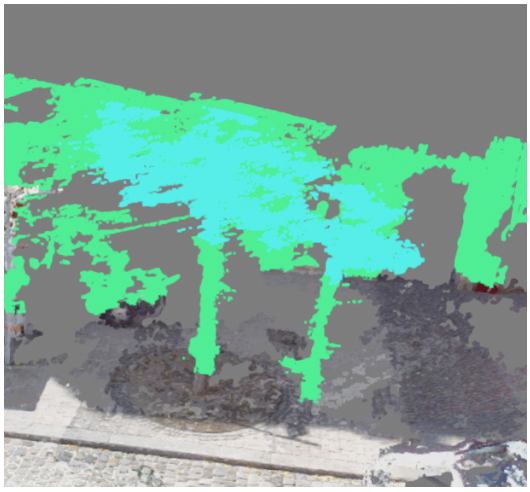

Figure 3.4: Example of a clustering result with the top & bottom of a tree being separate clusters

Figure 3.5: Example of optimal clustering result where the whole tree belongs to one cluster

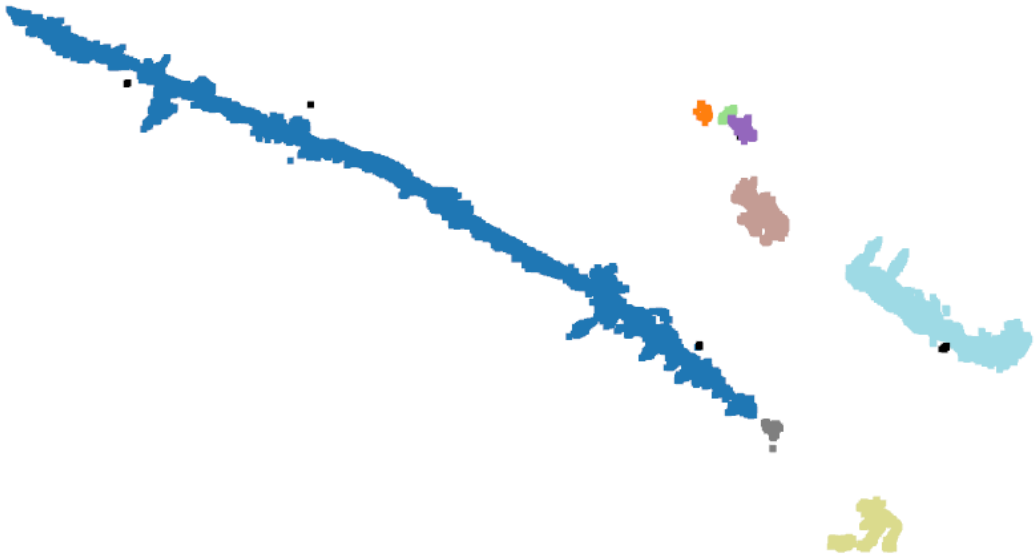

Figure 3.6: Clustering of building class, points in black are noise points

make another one worse so it is much easier to assign noise points if they are within an existing cluster. Examples of different 'eps' values and the noise points can be found in the 'Evaluation' chapter.
Since DBScan is run for each class and only returns an array of integers that represents the cluster values for each input point, a method for distinguishing the different classes and their clusters is necessary. In order to do that the cluster values are converted to float numbers where the number before the decimal point is the cluster number. The point after the decimal point is the class value. For example cluster value 3.5 would be 'Vegetation 3' because 5 represents vegetation and 3 is the cluster number. The reason why the cluster values are converted to float numbers and not to strings or any other type is that only integers and float numbers are allowed to be added to a laz dimension and as already mentioned that is how the cluster values are stored. After running the above-mentioned method for each of the split parts of the cloud, all of them are combined by stacking the results into one array and then storing them in a laz file. The benefit of creating a laz file is that a new dimension 'clusters' can be added and then afterwards be read with the '.clusters' attribute to obtain the cluster values.

3.3 Operations

After clustering a point cloud it can be opened in our program. Other file types than laz can also be used but those cannot save the clustering results due to their structure not allowing to add of a new attribute for the cluster values.
A clustered point cloud can be filtered by one of the following 6 classes which were already mentioned at the beginning of this chapter: building, car, ground, pole, vegetation as well as human. Human is not a class that can be assigned by the pretrained GRanD-Net model but since our dataset occasionally contains some, it can be done manually through our program. By choosing to filter a class, all clusters of the given class are colored differently and a label above each cluster's center with the class and cluster number as its value is placed. So when filtering a cloud by 'car' above the points that have the cluster value '1.2' the created label is 'Car 1' as 1 is the cluster number and 2 is associated with the 'car' class. The label is placed above the cluster's center as that seems to be the best place visually to match a label with its cluster. In order to prevent from labels being placed within the point cloud the biggest z value of the whole cloud is evaluated once before placing the labels since that value does not change. The center of the cluster is calculated by calling the open3d function 'get_center()' from the PointCloud Class [10]. The function is based on the C++ 'ComputeCenter()' function [2]. It simply adds up all the points of the cloud and then divides the sum by the number of points. In order to place the labels the function 'add_3d_label()' of the SceneWidget Class is called [12]. With that approach, each label is placed at the center x and y values of its cluster with the z value being the highest z value of the whole cloud. An example can be found in Figure 3.7

As briefly mentioned, a whole cloud or parts of it can be clustered manually by creating an 'OrientedBoundingBox' with open3d [9] which can be modified. After creating a box all of the points inside are marked in green, specifically RGB value (0, 1, 0) as displayed in Figure 3.8.

To evaluate which points are inside open3d's method 'get_point_indices_within_bounding_box' is called with the points of the loaded point cloud. Those points can also be filtered by one of

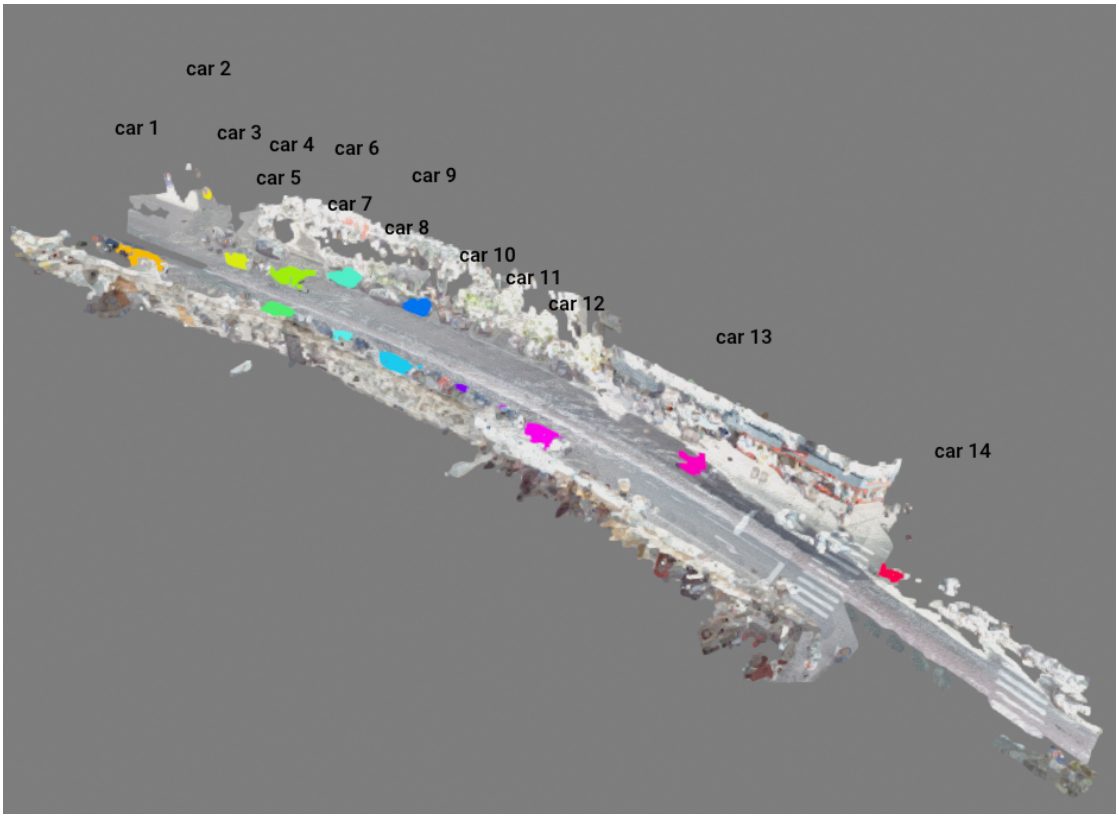

Figure 3.7: Example of how clusters specifically of the 'car' class are shown in our software

the classes. In that case, the points that belong to the chosen class are marked in green whereas the other ones are colored in their original color. An example can be found in Figure 3.9.

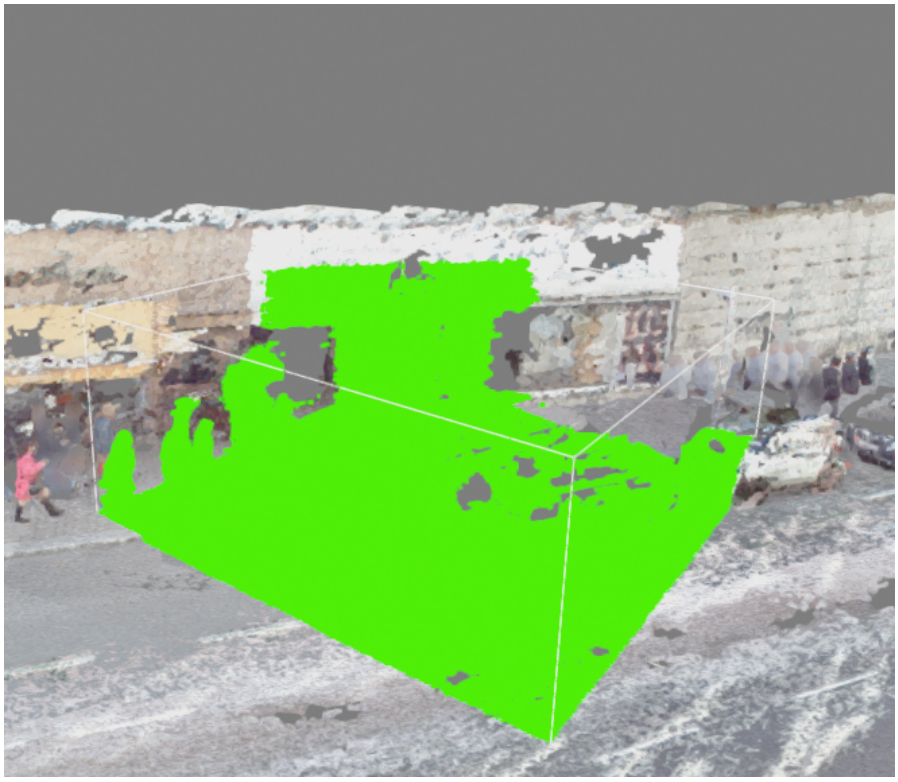

Figure 3.8: Example of created bounding box, points inside are colored in green

Figure 3.9: Example of the same bounding box as in Figure 3.8 but with 'car' class selected as filter

If a class is not represented in the points, nothing will be marked in green. Furthermore, the marked points can be assigned to a different cluster and class than their current one by opening the 'Classification' menu and clicking 'Set [desired class]'. Additionally, there is an option to only select existing clusters. In that mode, only points of the selected object can be changed. So if one wants to change the class of a cluster or parts of it, they can select the cluster via 'Select Object' and then modify the box accordingly in order to assign another class for the marked points. An example of how that can be done is displayed in Figures 3.10 to 3.13.

Since open3d's PointCloud Geometry [10] does not have a cluster or classification attribute, the cluster values are stored in a separate array where the indices of the values align with the order of the points. That obviously means that after modifying parts of the cloud, the cluster array has to be adjusted accordingly when removing points or changing the indices of some points in the cloud.

In order to have no restrictions regarding the selection of points, the box can be modified through translation, scaling, and rotation. All of those operations are done with open3d's methods for those transformations (translate(), scale(), rotate()). The default translation is on the x- and y-axis plane. Translation can also be limited to one of the 3 axes (x, y, z). The default scaling operation is in all axes but just like translation, scaling can be limited to one of the 3 axes. Rotation, however, is only available on the z-axis in order to not overload that feature and keep it simple for users. An example of how to move a car with a box created with the 'Select' operation is illustrated in Figures 3.14 to 3.17. All of the previously mentioned transformations are necessary for that.

Once the box is placed and the desired points are marked, those can either be moved to another location or deleted from the cloud.

Since GRanD-Net, as well as the clustering algorithm mentioned in the previous section, tend to have misinterpretations and do not always deliver optimal results as displayed a couple of times in this chapter, our program can be used to manually fix these mistakes which was also shown.

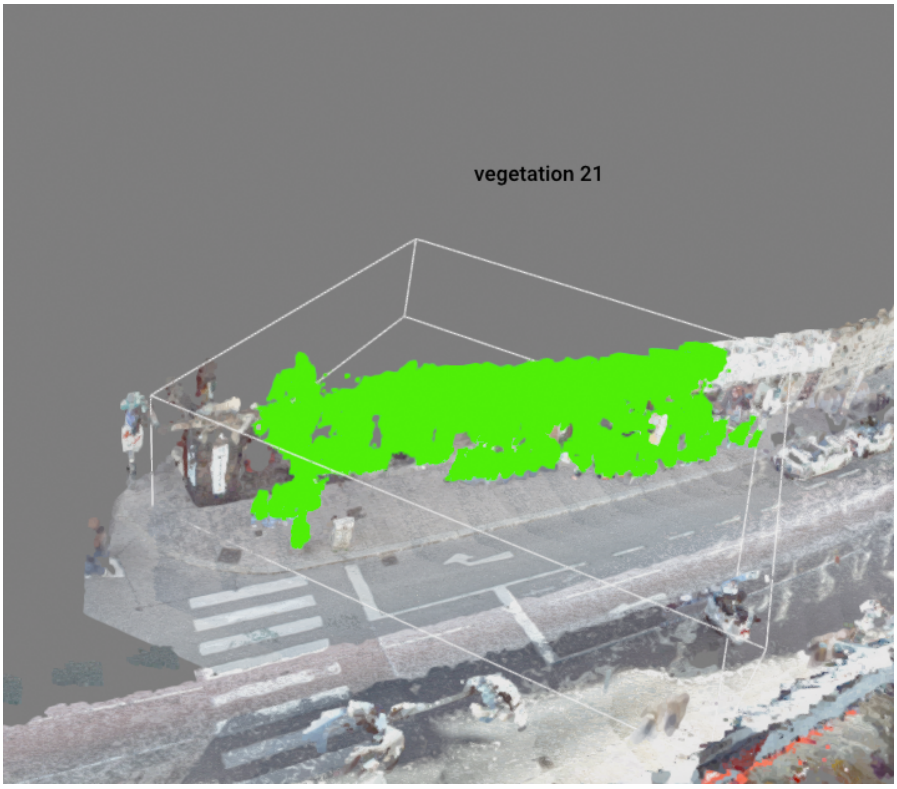

Figure 3.10: Example of misclassification where a building is classified as vegetation

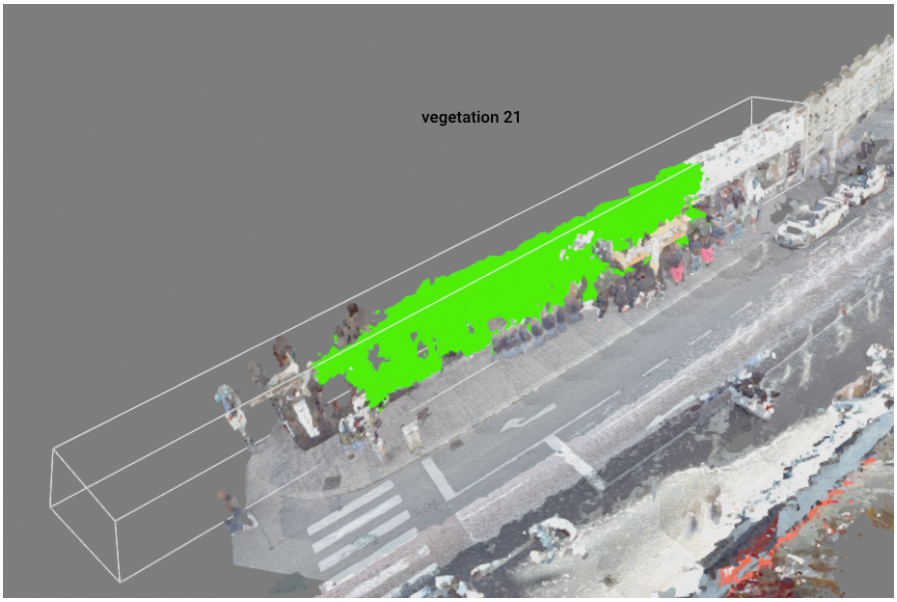

Figure 3.11: Adjusting the box to the building part of the object

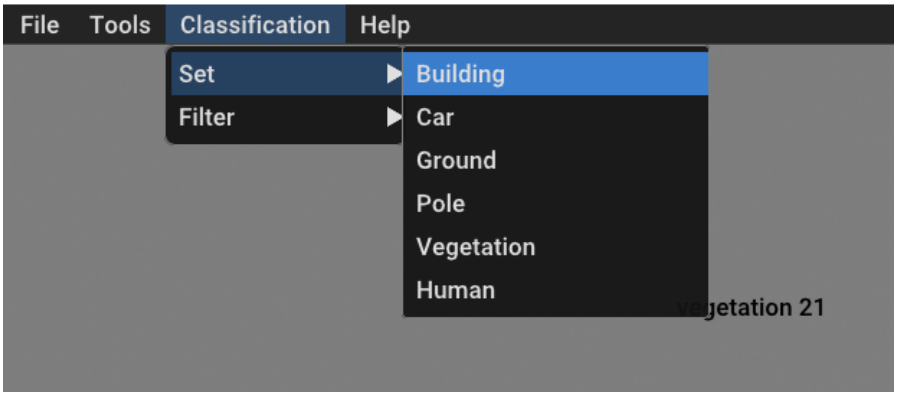

Figure 3.12: Select building operation

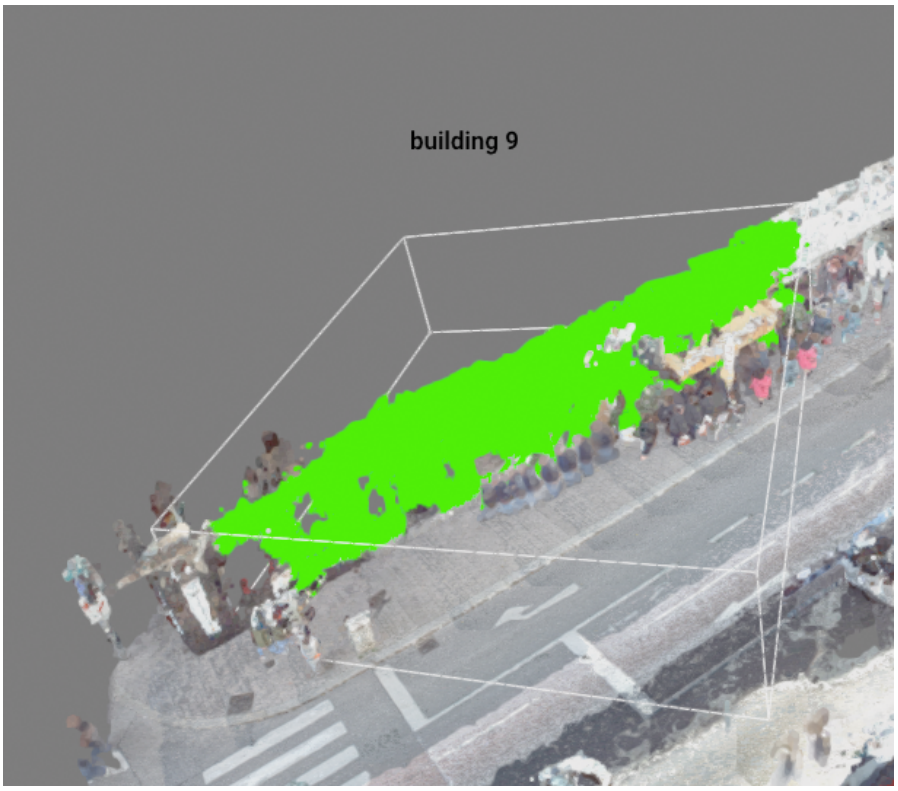

Figure 3.13: Selected same area as in Figure 3.10, now it is classified correctly

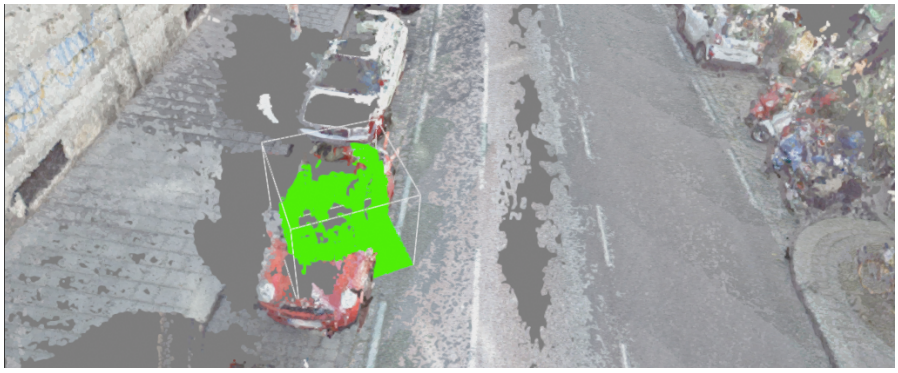

Figure 3.14: Clicking on an area with a car after selecting the 'Select' operation creates a bounding box

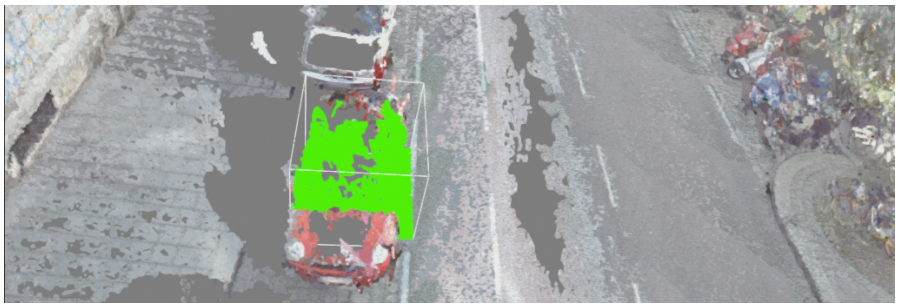

Figure 3.15: The bounding box is rotated with key 'R' to fit the car

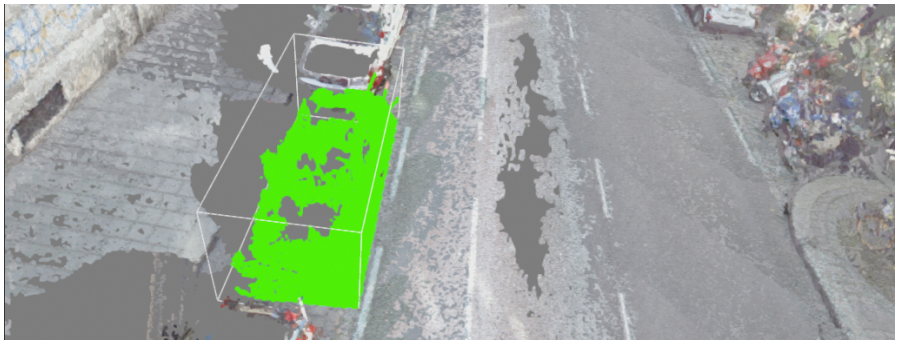

Figure 3.16: The bounding box is scaled in the x or y direction by clicking 'S' and then either 'X' or 'Y'

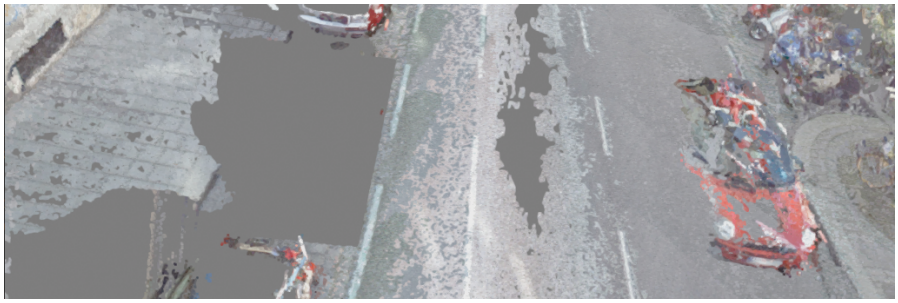

Figure 3.17: The points inside can be moved with key 'M'

CHAPTER 4

Evaluation

In this chapter different aspects such as results of the classification algorithm, GRanD-Net, [13] and run-time for the various operations that are mentioned in the 'Methodology' chapter were evaluated on our dataset. Additionally, the self-written scripts that were mentioned in chapter 3.1 were tested with the SHREC 2020 dataset [1] as the ground truth is available for those which was used to figure out whether those scripts might be beneficial for another dataset.

4.1 Hardware & Software Specification

My colleague Manuel Keilman and I decided to use Python 3.9 which at the time of developing the program was the latest version that supported the open3d library which was the one we mainly used [8]. The software was tested on Windows 10 as well as Linux Ubuntu 22.04. The classification algorithm however was only tested on Linux Ubuntu 22.04.
The following Hardware was used for the 'Classification' section:

- CPU: AMD Ryzen 7 5800X, 8-Core, 3.8 GHz
- GPU: NVIDIA GeForce RTX 3080
- RAM: 64.0 GB
- OS: Linux Ubuntu 22.04

The following Hardware was used for the 'Clustering' and 'Program / GUI' section:

- Notebook: Asus UX550VE
- CPU: Intel Core i7-7700HQ, 4-Core, 2.8 GHz
- GPU: NVIDIA GeForce GTX 1050 Ti (4GB VRAM)
- RAM: 16.0 GB
- OS: Windows 10 Home, 64-bit

4.2 Classification

In this section, the performance and the results of the GRanD-Net algorithm will be reviewed in detail. Performance, in this case, equals the run-time to classify point clouds with different amounts of points. Since there is no ground truth information available for our dataset, the classification results will be judged by the visual appearance of the classified point clouds. Due to voxel subsampling being an option the clouds will be subsampled with different factors and it will be evaluated whether that improves the classification results. As mentioned our dataset will be judged by its visual appearance. The SHREC dataset, however, contains the ground truth, which is why the subsampling will also be evaluated on that dataset in order to see the impact on another dataset.

SHREC Dataset

In this section, only the voxel subsampling script is evaluated since the results of the GRanD-Net algorithm on this dataset were already reviewed in [13]. Out of the 60 available test point clouds [1], the following ones were randomly picked: 5D4L1TX7.txt in Figure 4.1, 5D4KVQ9U.txt in Figure 4.3 and 5D4KX3TQ.txt in Figure 4.2.
All of those point clouds were classified with a couple of different input values and without subsampling in order to see the impact of the method. Despite subsampling reducing the total amount of points of cloud, the GRanD-Net algorithm took just as much time as without subsampling. The average run-time was 19 min and 12 sec.
After running the GRanD-Net algorithm on each of these files the created label file by GRanD-Net which was mentioned in chapter 3.1 was iterated through and then compared to the ground truth of the point clouds. Since the ground truth was given as the 4th column of the mentioned txt files, the values of the 4th column were iterated through and then compared to the created label file. For each line that matched a counter was incremented by 1. In order to obtain the total percentage of correctly classified points, the counter was divided by the total amount of points. As shown in each of the graphs the subsampling method performed way worse than not subsampling at all. In general, the more is being subsampled the worse the result ended up being. Even removing roughly 0.3 % of the points for 5D4KX3TQ resulted in a 42% worse result as seen in Figure 4.2. As shown in all of the graphs it is not worth subsampling the point clouds of the SHREC dataset at all, especially since the run-time is not reduced. A reason why the SHREC dataset performed much worse than our dataset could be that the clouds in the SHREC dataset are not as dense regarding points. The points are spaced out more and therefore creating voxel-grids could end up removing relevant information for the classification algorithm. In our dataset, however, the clouds are rather dense and therefore combining points could end up with a more clear model.

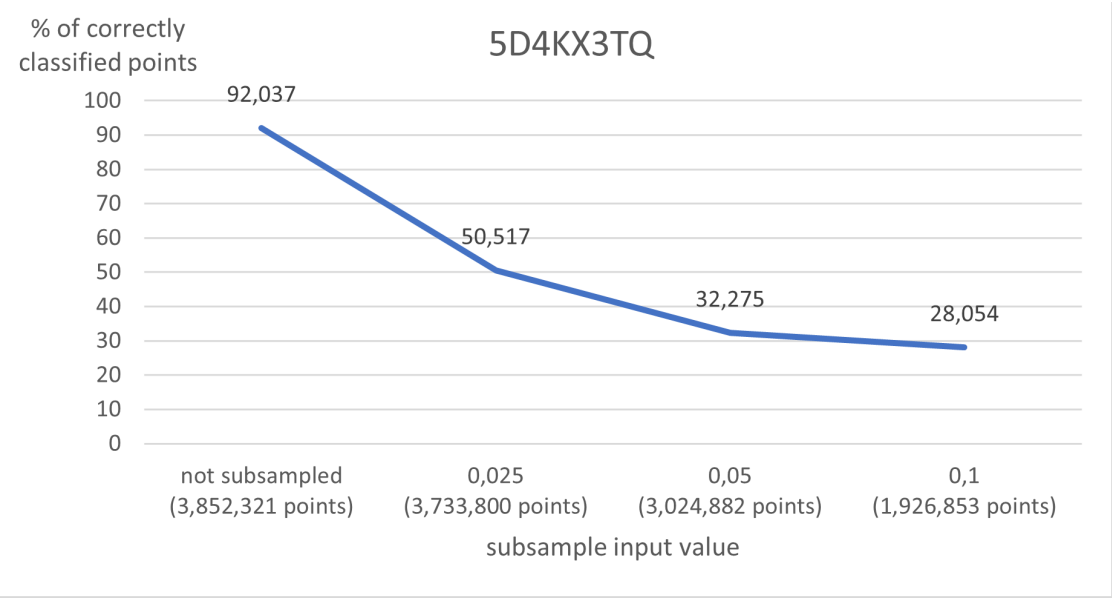

Figure 4.2: Percentage of correctly classified points with different subsampling input values for 5D4KX3TQ.txt. The subsample input value is multiplied by 0.2 and the result of that is the size of a grid cell. For example, 0.2*0.5 ends up with a grid cell size of 0.1.

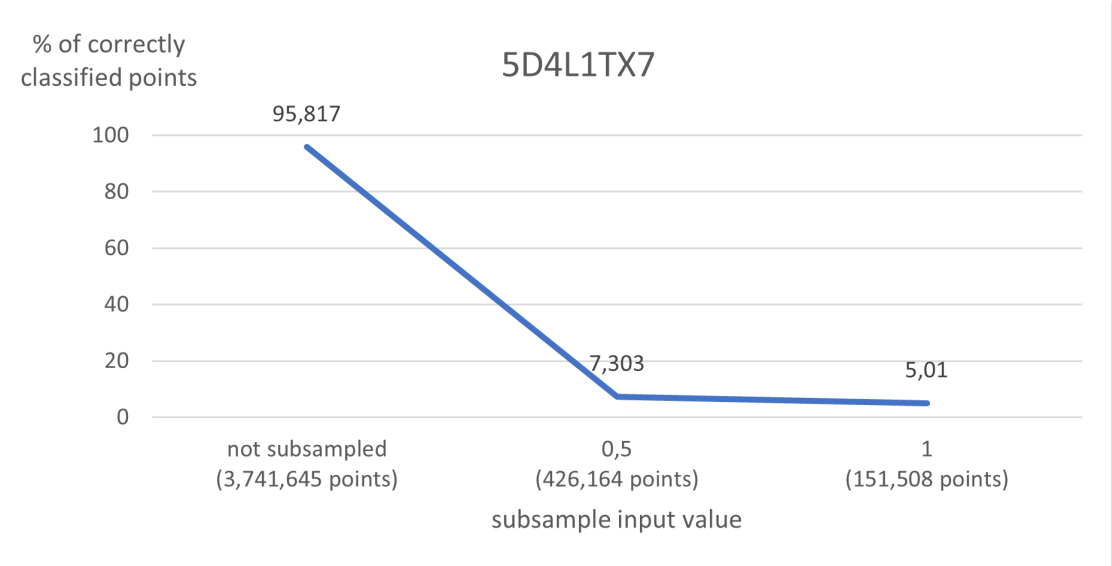

Figure 4.1: Percentage of correctly classified points with different subsampling input values for 5D4L1TX7.txt. The subsample input value is multiplied by 0.2 and the result of that is the size of a grid cell. For example, 0.2*0.5 ends up with a grid cell size of 0.1.

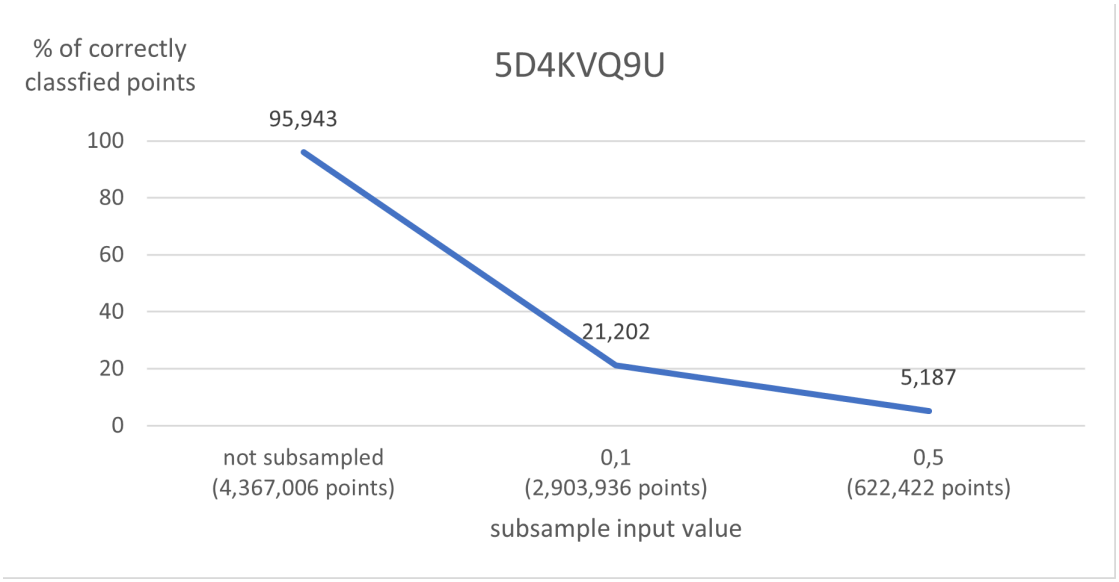

Figure 4.3: Percentage of correctly classified points with different subsampling input values for 5D4KVQ9U.txt. The subsample input value is multiplied by 0.2 and the result of that is the size of a grid cell. For example, 0.2*0.5 ends up with a grid cell size of 0.1.

Our Dataset [6]

In this section, the following point clouds were classified with and without subsampling: 8088.laz 4.4, 741.laz 4.5, 8090.laz 4.6 and 33476.laz 4.7. Since the ground truth of the classes for each point is not available, the result of the classification will be judged by its visual appearance. If for example a car is mainly classified as something else then that is obviously a suboptimal result. Depending on how much the original cloud is subsampled it can increase the percentage of the points of the car that are identified properly which will be discussed further later on in this section. Even though the point clouds consist of different numbers of points the run-time for the GRanD-Net algorithm remained the same which was about 19 min 10 sec just like with the SHREC dataset. In order to visualize the classes assigned to a point cloud the script, 'visualize_laz_classification.py' can be used after running the 'add_labels_to_laz.py' script which was mentioned in section 3.1. In the 'visualize_laz_classification.py' script the color scheme works as follows: buildings are blue, cars are red, the ground is brown, poles are yellow and vegetation is green. Examples of what the result looks like can be found in Figures 4.8 to 4.19.

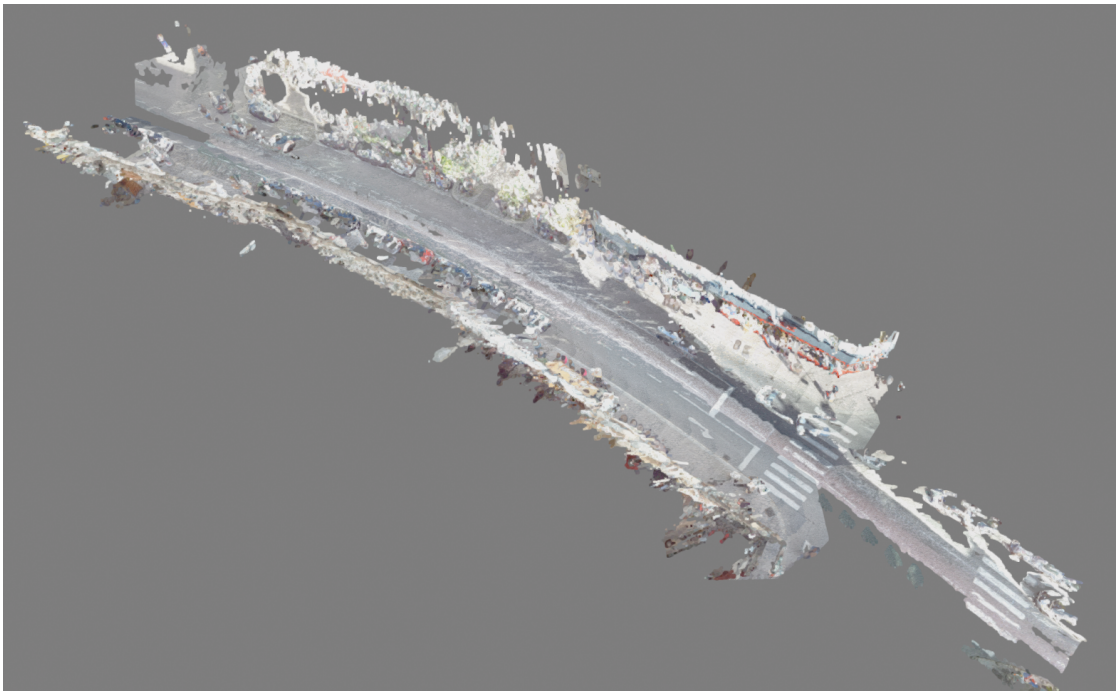

Figure 4.4: Point cloud 8088 of our dataset.

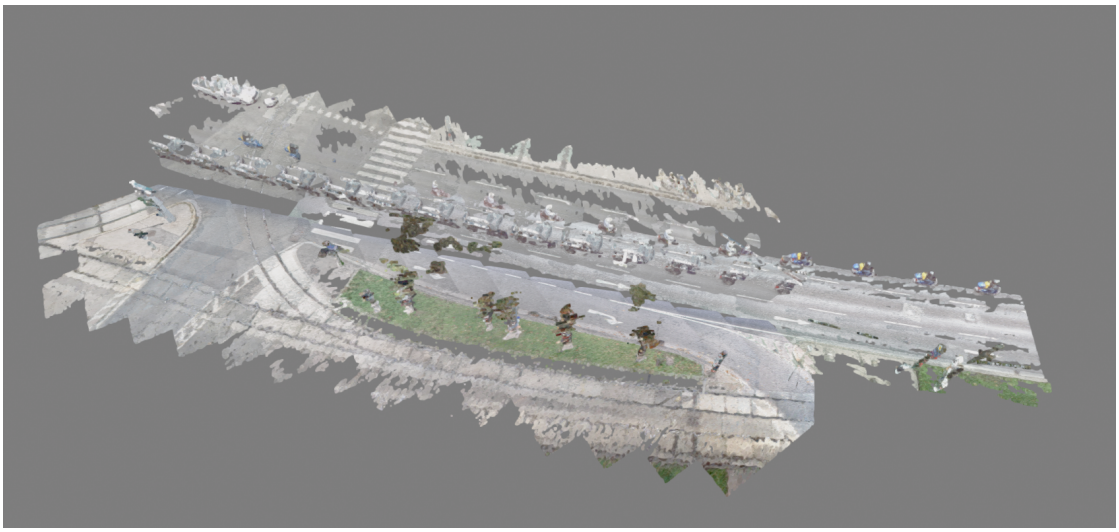

Figure 4.5: Point cloud 741 of our dataset.

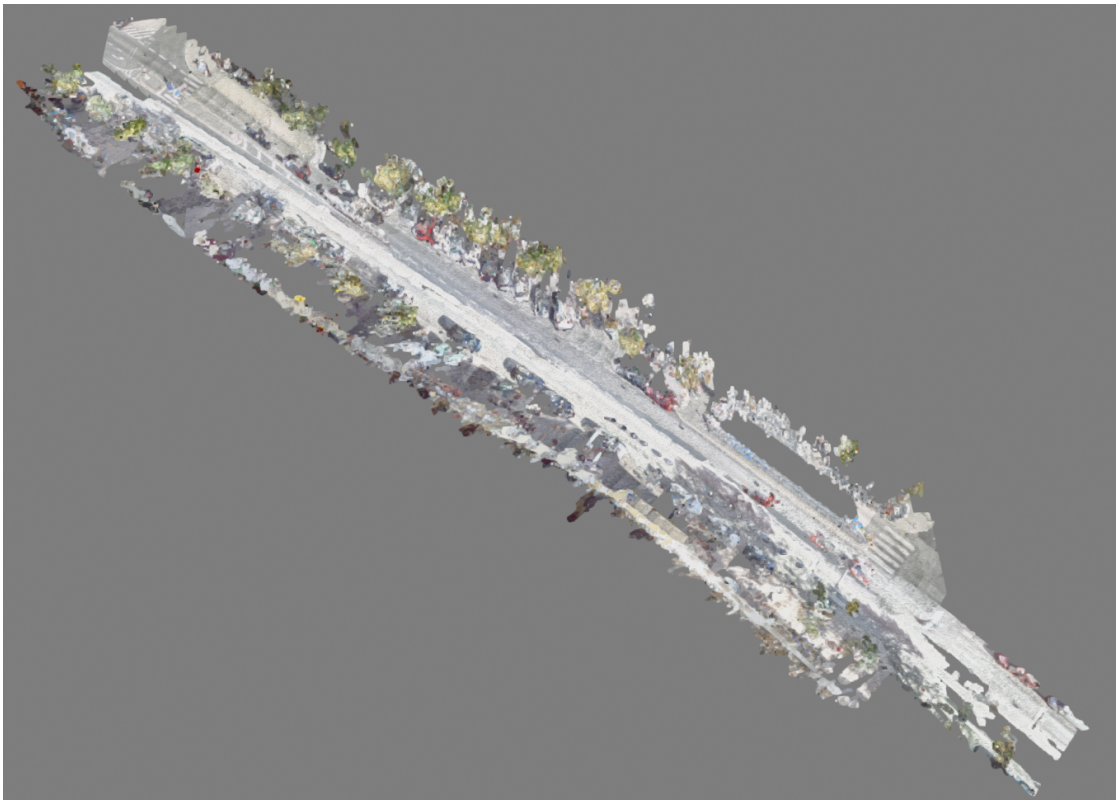

Figure 4.6: Point cloud 8090 of our dataset.

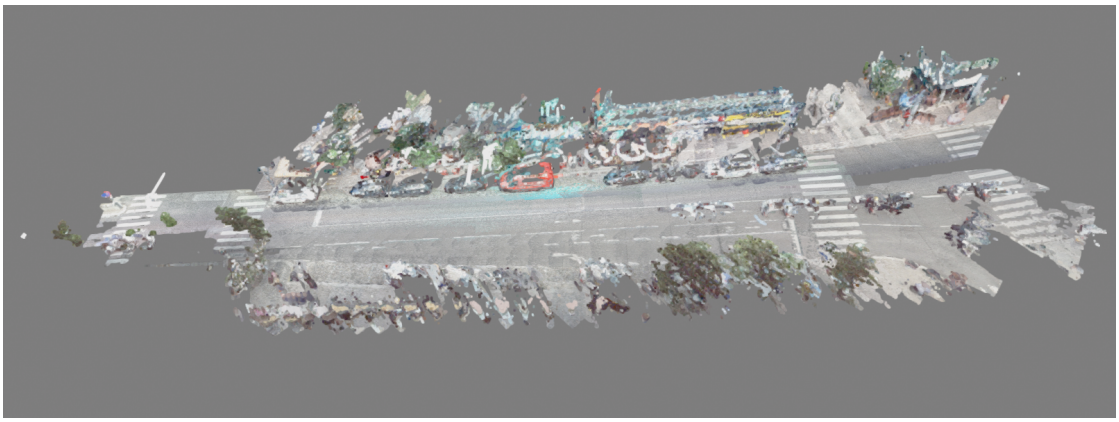

Figure 4.7: Point cloud 33476 of our dataset.

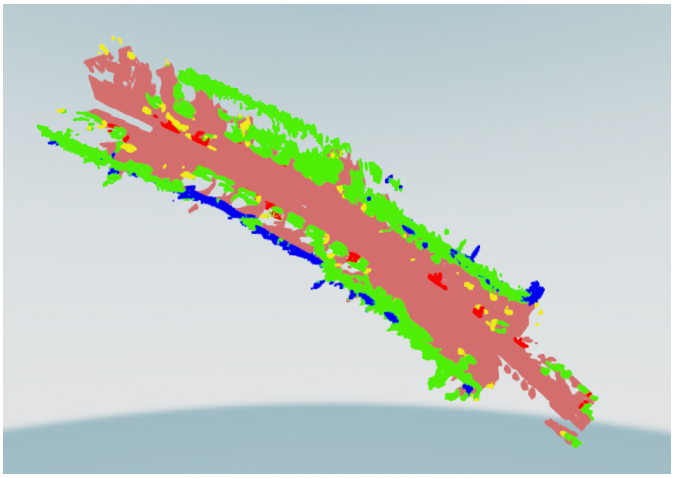

Figure 4.8: Classification of 8088.laz not subsampled. As displayed the cars at the center and the majority of the buildings are misclassified as vegetation.

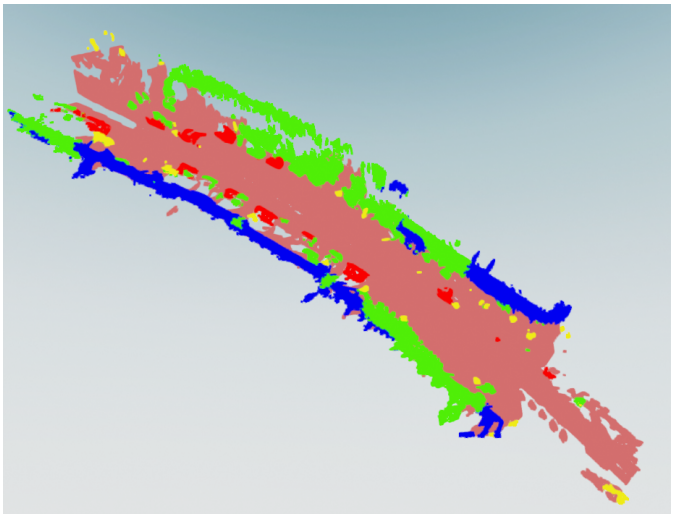

Figure 4.9: Classification of 8088.laz subsampled with a factor of 0.5. Subsampling the cloud improved the result compared to Figure 4.8. The majority of the cars at the center are classified as such. More parts of the building are classified properly as well.

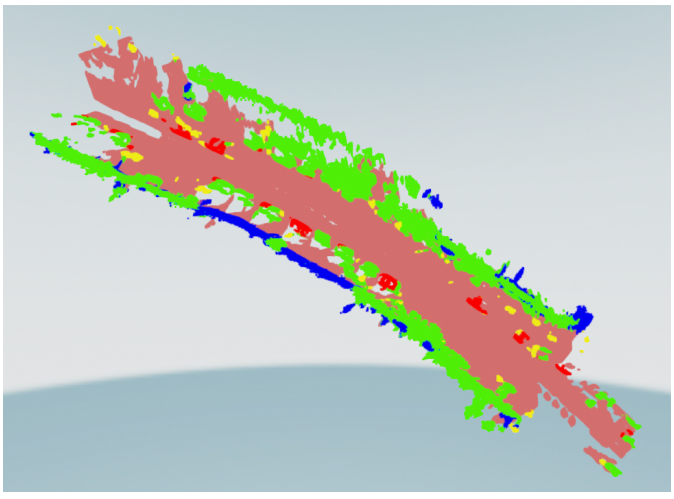

Figure 4.10: Classification of 8088.laz subsampled with a factor of 0.25. Subsampling even further ends up with a similar result as in Figure 4.8

.

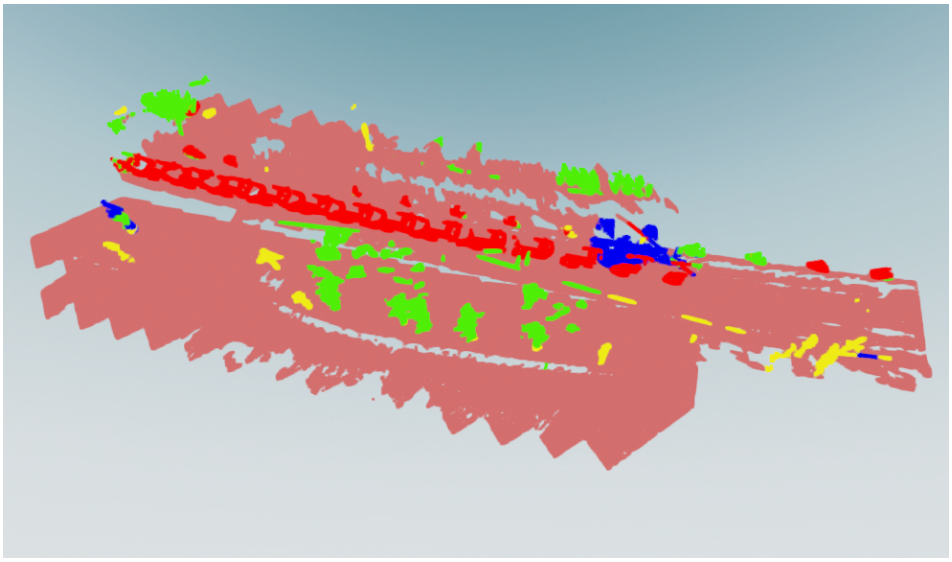

Figure 4.11: Classification of 741.laz not subsampled. As displayed the pole on the left is classified as a building and vegetation. Furthermore, the ground at the top is classified as vegetation.

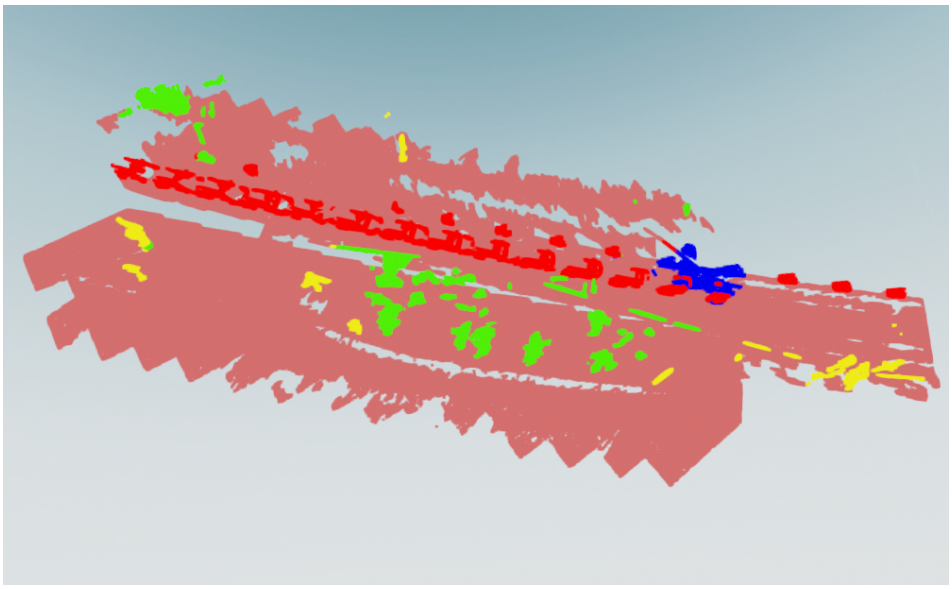

Figure 4.12: Classification of 741.laz subsampled with a factor of 0.5. Compared to Figure 4.11 the ground at the top is classified as such. Besides the pole on the left is identified properly.

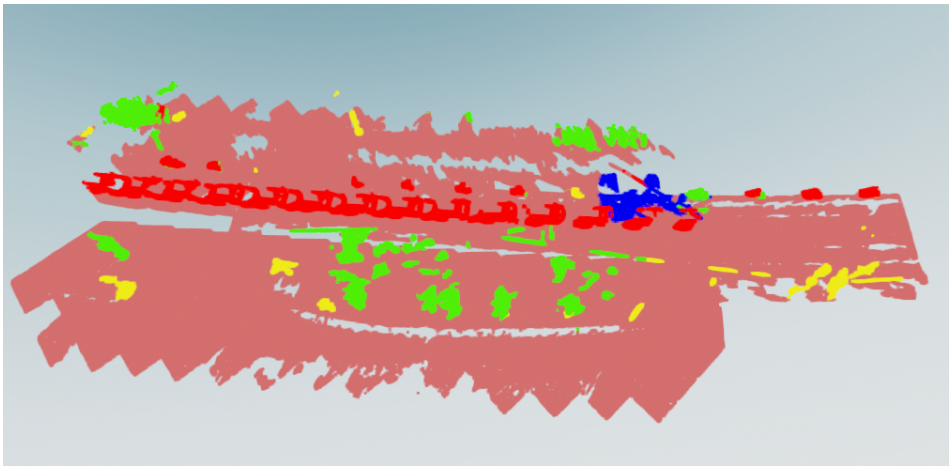

Figure 4.13: Classification of 741.laz subsampled with a factor of 0.25. Subsampling too much ends up with similar misclassification as in Figure 4.11.

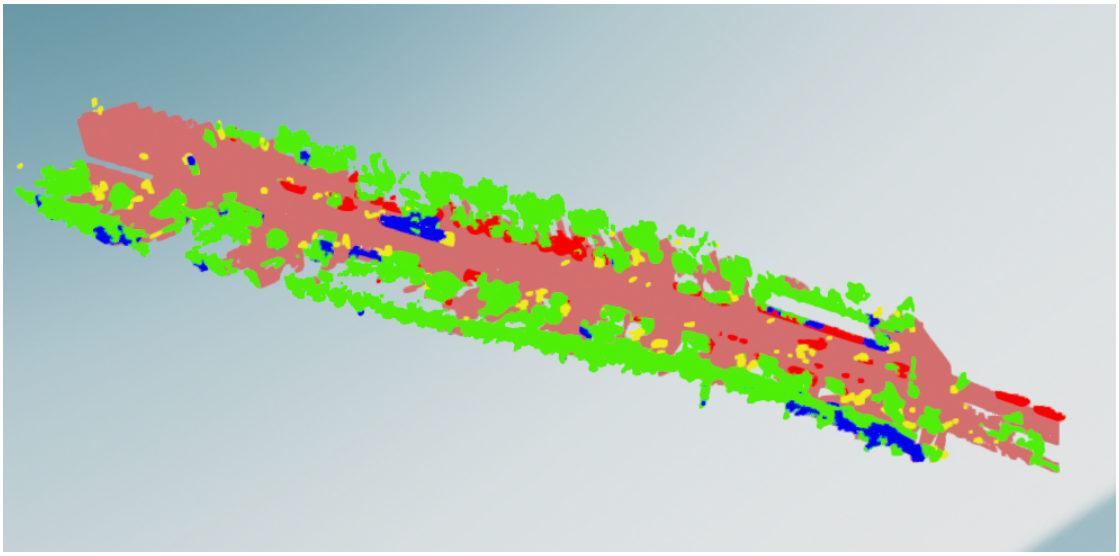

Figure 4.14: Classification of 8090.laz not subsampled. As displayed a lot of cars and buildings are classified as vegetation.

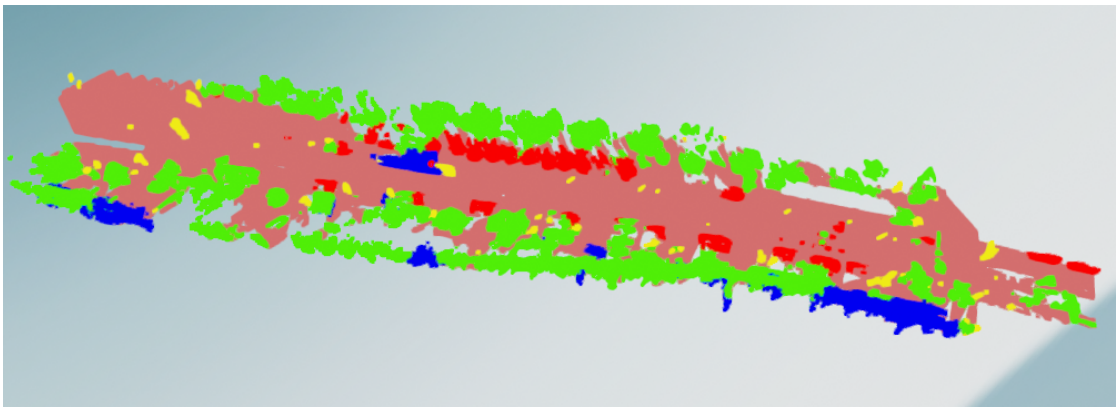

Figure 4.15: Classification of 8090.laz subsampled with a factor of 0.5. Subsampling the cloud ends up with more parts of the buildings being classified properly. Furthermore, the cars at the center are identified more as well.

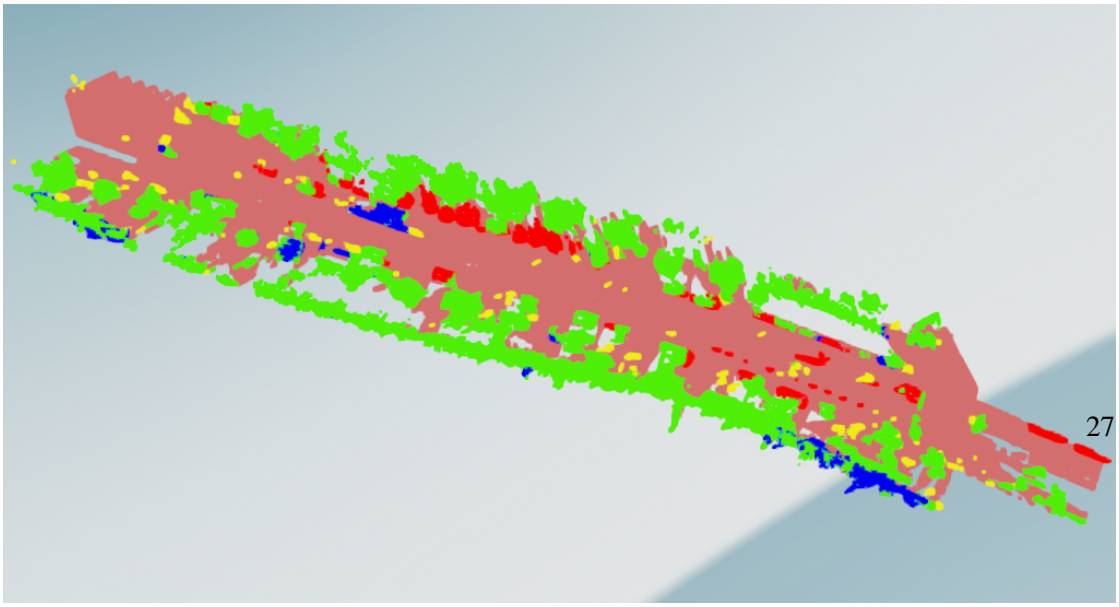

Figure 4.16: Classification of 8090.laz subsampled with a factor of 0.25. Subsampling more than in Figure 4.15 ends up with similar miclassification as in Figure 4.14.

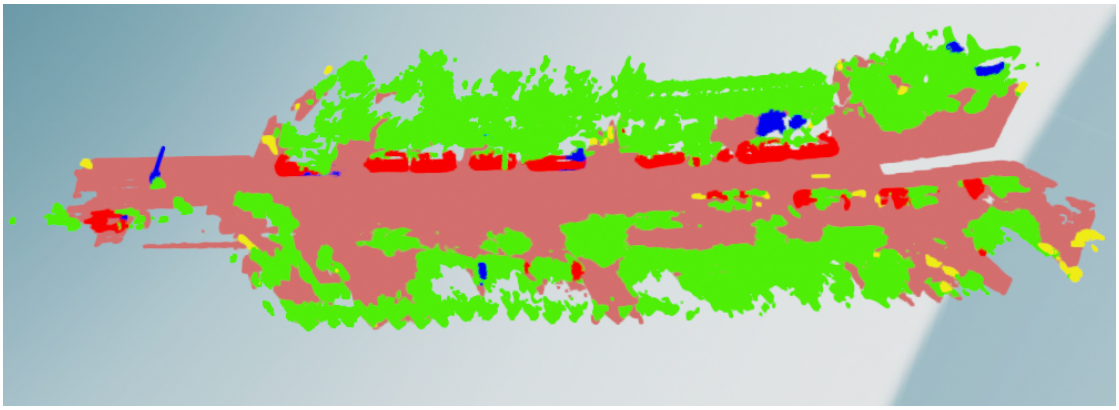

Figure 4.17: Classification of 33476.laz not subsampled. As displayed some cars on the left side are classified as cars and vegetation. Besides the pole on the right is identified as a building and vegetation.

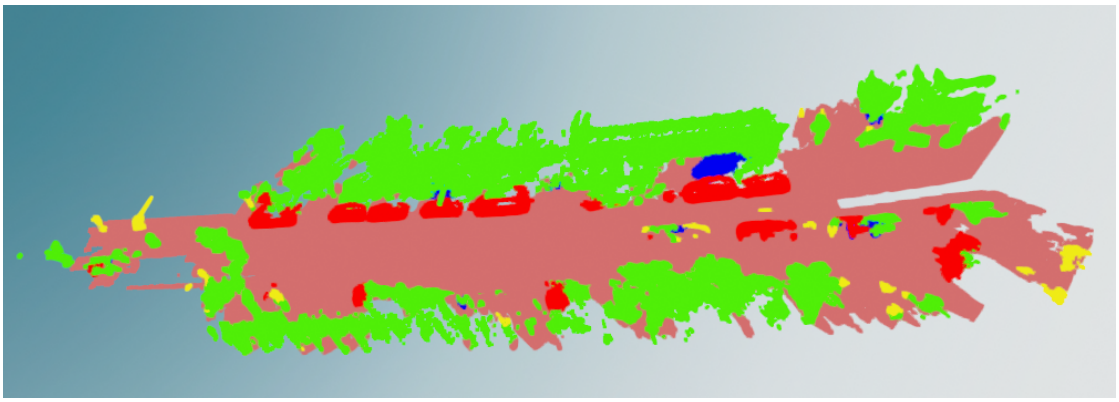

Figure 4.18: Classification of 33476.laz subsampled with a factor of 0.5. Compared to Figure 4.17 the pole on the left is classified as such. One car on the left is identified properly as well.

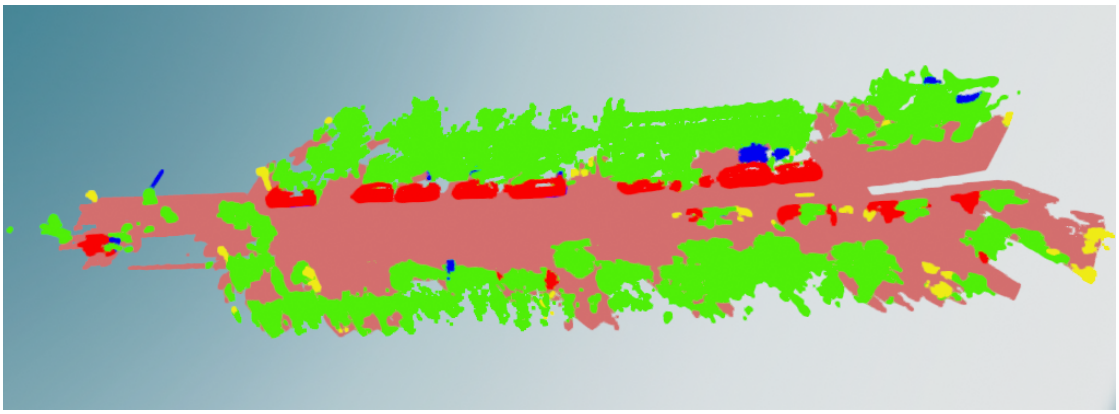

Figure 4.19: Classification of 33476.laz subsampled with a factor of 0.25. Subsampling more than in Figure 4.18 ends up with similar results as in Figure 4.17

As it is shown in the Figures 4.8 to 4.19 the classification overall seems better when subsampling the clouds. The value with the most optimal results of the ones that were used seems to be 0.5. As seen in Figures 4.8, 4.10 for cloud 8088 and 4.14, 4.16 for cloud 8090 some cars at the center of the clouds were not classified as such but rather as ground or vegetation. However, those are classified more accurately when subsampling the cloud with a value of 0.5. Another example would be that in Figures 4.17, 4.19 for cloud 33476 and 4.11, 4.13 for cloud 741 the pole on the left side of the image is classified as a combination of building and vegetation. In Figure 4.18 and 4.12 however, the pole is completely classified as such. Even though the results are overall better certain areas are still classified almost completely wrong. The most frequent misclassification is that buildings are classified as vegetation. Hence, the option to manually classify those parts in our software. Another reason is that humans are not recognized at all since that is not an option with the GRanD-Net algorithm. Overall the results are rather good for some clouds as seen in Figure 4.12 where almost all the cars are classified correctly.

4.3 Clustering

In this section, the run-time to cluster the clouds 741, 8088, 33476, and 8090 with a subsampling factor of 0.5 is evaluated. Besides the visual appearance of the clusters is evaluated. Since DBScan is used which has the 'eps' value and the minimal number of points needed for a cluster as its parameters those will be reviewed. The 'eps' value determines if two points are within the same neighbourhood. If the distance between both is greater than the 'eps' value they are not considered in the same neighbourhood and therefore not in the same cluster.
The minimal number of points needed for a cluster does not have a lot of impact unless the number is quite big which usually does not end up with good results because a lot of areas are considered noise points. Noise points do not belong to a cluster since they do not fulfill the input parameters. A comparison between a parameter value of 175 points and 25 points for the points that were classified as vegetation can be found in Figures 4.22 and 4.26. It is clearly visible that setting the number for the minimal number of points too high ends up with a lot of noise points

The more impactful parameter, however, seems to be 'eps' since it determines what belongs to a potential cluster. In general the higher the 'eps' value the longer the run-time of DBScan but the results seem to be better in most cases. In the subsection 'eps', different 'eps' values are compared. All of them were tested on the point clouds 741 and 8088 with the same minimal number of points value.
Additionally, the difference between splitting the cloud into its classes before clustering and clustering the whole cloud at once is evaluated. In both cases, the ground is not clustered since there usually is one ground. While clustering everything at once is faster and might seem better at first, the fact that a connected component might contain multiple classes has to be considered. For example, a cluster might contain a wall but parts of the wall are classified as vegetation and they are not close to each other. In that case, the vegetation cluster could be far apart as displayed in Figure 4.20.

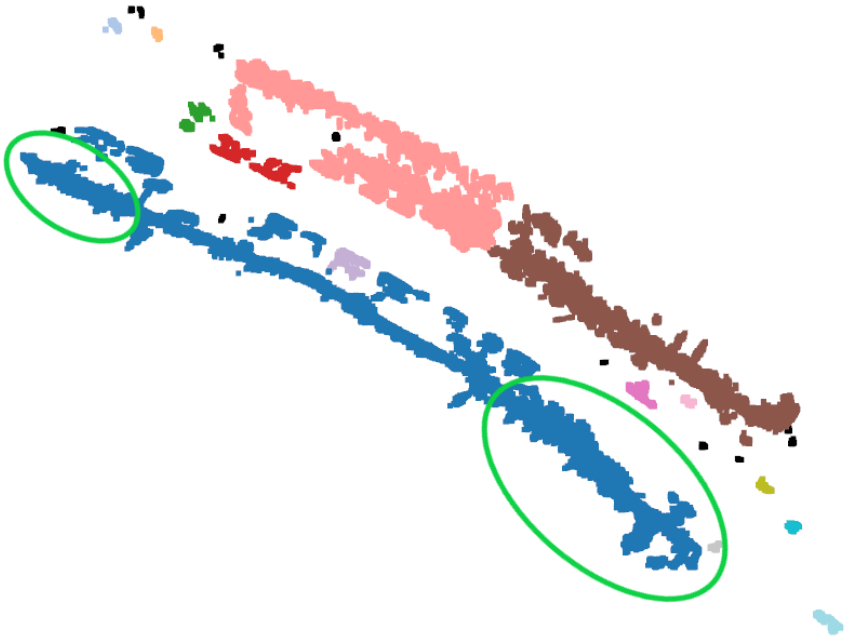

Figure 4.20: Clustering the whole cloud without splitting it into classes. Each color represents a cluster in this Figure. The classes for this cloud can be seen in Figure 4.9. Green circles show the vegetation part of the blue cluster. Both areas would be considered as one cluster which is suboptimal as they are far apart.

Minimal number of points

The minimal number of points required to be considered a cluster does have an impact on the results of the clustering but if the value is not too high it does not affect it a lot. Obviously, if the value is too high which for our dataset would be above 175 points there will be quite a lot of noise points which has an impact on how the clusters are put together. As seen in Figure 4.21 the majority of the cars are regarded as noise points and even for vegetation a lot of points are considered noise as displayed in Figure 4.22.

The results are much better when taking 100 points as a value but there are still quite some noise points as seen in Figures 4.23 and 4.24.

Therefore 25 was chosen as a value since the noise points for that value are rather small and not close to each other as displayed in Figure 4.25 and 4.26.

'eps'

The eps value does have more impact on how the clusters are formed than the min. points but the algorithm also takes longer when increasing that value. A high value like 1.75 ends up with the best results for cloud 8088.laz as displayed in Figure 4.28 compared to 4.27 which has an eps value of 0.5. However, a value of 1.75 is not as good for cloud 741.laz as displayed in Figures 4.29, 4.30 and 4.31. Since the range for points in which neighbors can be found is increased with a higher eps value the cars of 741.laz are all considered as one object whereas for eps value 0.5 there is a better distinction between the smaller parts. The reason a higher value is better for 8088.laz is partly because its parts such as the buildings are much bigger in general. So with a higher value, more points in the nearby area will be considered. Since the cars of 8088.laz are not near to each other the higher value does not negatively impact those clusters as it does with 741.laz. Choosing the right eps value is therefore important for having good clusters and a higher value does not always end up with better results.

Splitting into classes vs all at once

Splitting the cloud into its different classes ends up with better clustering results compared to clustering the whole cloud at once. Clustering everything at once is much faster since the algorithm only has to be applied once. Clustering the cloud 8088 with 'eps' value 1.75 and min. points 25 took 10 seconds less when applying the algorithm to the whole cloud. Since clustering is only performed once before using the application the run-time is not as important as the final results. As displayed in Figure 4.20 the clusters of clustering all at once might seem good enough but it does not take into consideration which classes are in a cluster. Even though the blue building for 8088.laz is one cluster that is almost ideal not all of those points are classified as such. Therefore splitting that cluster into its classes afterwards can end up with clusters not being a unit but rather parts that are far apart which in the end is not a good result.

Overall run-time

The run-time of the whole clustering process took between 17 and 44 seconds where the number of points of a cloud and the eps value had an impact on that time. Cloud 741.laz has about 2.12 million points and with eps 0.75 it took 17 seconds. The same cloud with eps 1.25 took 28.3 seconds and with eps 1.75 even 44 seconds. The eps value, however, does not always have such an impact as cloud 8090.laz which has 9.34 million points took 37.4 seconds with eps 0.75 and only 38.29 seconds with eps 1.75 which is only a minor increase.

4.4 Program / GUI

In this section, the run-time, as well as the visual appearance of the following operations of our software, are evaluated: the classification 'Set [class]' as well as the 'Filter [class]' options. Those operations were applied on the 8088.laz cloud which was classified with a 'voxel_size' input of 0.5 and clustered with an 'eps' value of 1.25 and min. points value 25.

Run-time

The average run-time for the 'Set [class]' operations was between 0.2 and 0.25 seconds. The bigger the area the longer it took but since it took a quarter of sec at max it is a rather fast operation. The reason for that is that the operation does not require rendering the scene again. When setting a class the indices of the points within the created box are retrieved. Afterwards those indices in the designated arrays for the cluster and class values are changed. The new cluster value of a class is always the current max of that class incremented by one.
'Filtering [class]', however, took much more time compared to the 'set' operation because by filtering the clouds have to be rendered once to apply the changes which is not the case when setting a class. As displayed in Figure 4.32, it can take quite a few seconds to show the results. In general, if a class contains more points the longer it usually takes to render it. Since the ground usually contains the most points by far it takes the longest time when it is selected with the filter operation but since the ground usually is one cluster it is much more efficient to simply select it with the 'Select Object' operation as that takes about a third of the time filtering does.
When placing all 3 clouds into one scene it can take up longer than when summing up the times of each cloud individually, e.g. vegetation takes 17.25 seconds if all clouds are in a scene, but if each cloud is in a single scene it and then the time is summed up it takes 3.15 seconds less. Even though it might seem quite a bit that it takes 26 seconds for filtering the ground all of the clouds together contain about 20 million points of which the ground is about 11.5 million.

Visual appearance

In terms of visual appearance, there is nothing to say about the 'Set [class]' operation since that only changes the label by deleting the old one and creating a new one when a cluster is selected with the 'Select Object' operation.
In terms of filtering the cloud based on the classes, it works rather well as displayed in Figure 4.33 and 4.34. In general, the fewer clusters in the scene the easier it is to have a good overview which means that is clearly distinguishable which cluster belongs to which part of the cloud. If a lot of clusters for example 200 are in a scene there will be a lot of labels and therefore a lot of overlapping text which can be distracting. That however should not be a problem when zooming into specific areas. Another downside is that when a lot of clusters exist the difference in color at some point will not be noticeable as easily since a colormap is used which contains a limited amount of colors. Despite interpolating between those colors the difference is sometimes hard to notice. In Figure 4.34 'car 29' and 'car 30' almost have the same color but since the label is always above the center of the cloud users should still be able to figure out which label belongs to which car. In the worst case, they can use the 'Select Object' operation to figure out what the exact cluster is.

Figure 4.21: Clustering cars of cloud 8088 with a minimal number of points value of 175. Noise points are colored in black

Figure 4.22: Clustering vegetation of cloud 8088 with a minimal number of points value of 175. Noise points are colored in black

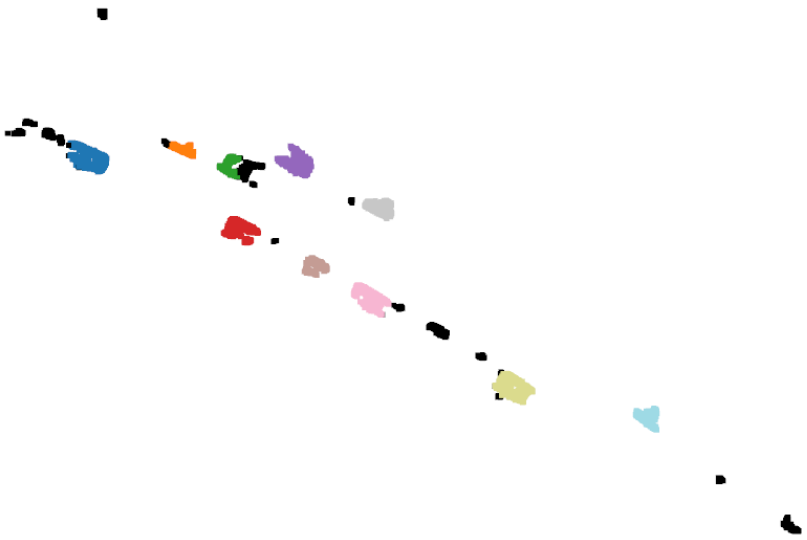

Figure 4.23: Clustering cars of cloud 8088 with a minimal number of points value of 100. Noise points are colored in black

Figure 4.24: Clustering vegetation of cloud 8088 with a minimal number of points value of 100. Noise points are colored in black

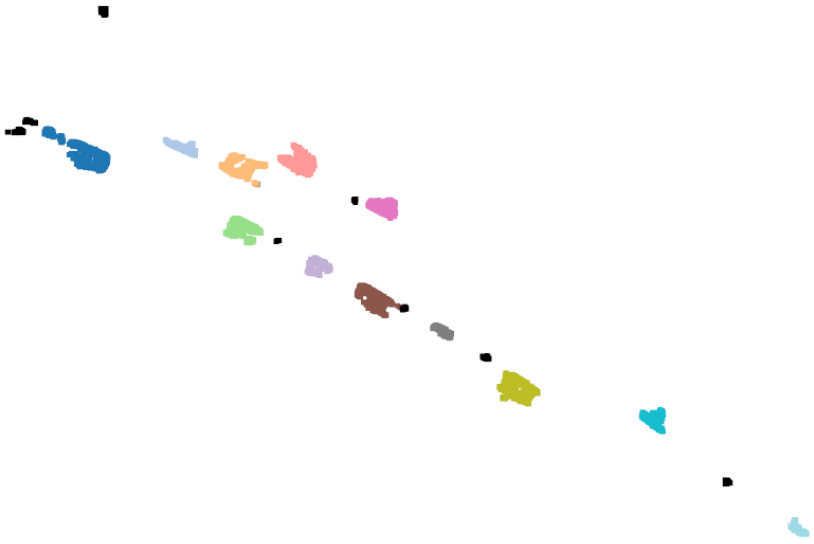

Figure 4.25: Clustering cars of cloud 8088 with a minimal number of points value of 25. Noise points are colored in black

Figure 4.26: Clustering vegetation of cloud 8088 with a minimal number of points value of 25. Noise points are colored in black

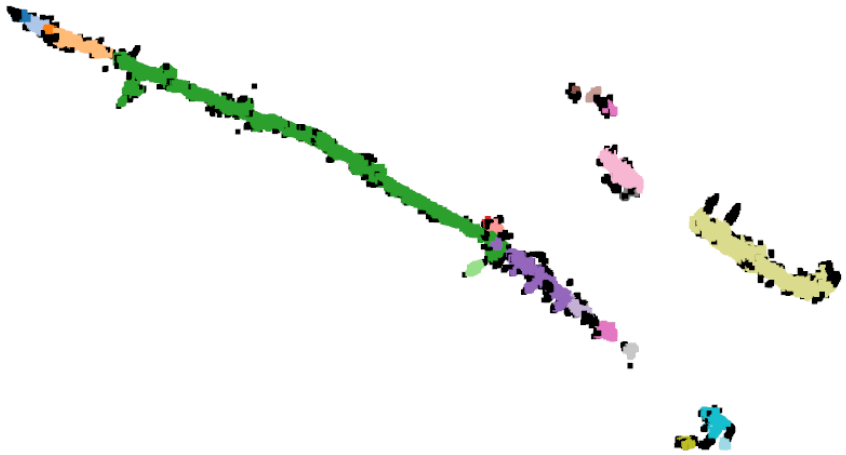

Figure 4.27: Clustering buildings of cloud 8088 with eps value 0.5. Noise points are colored in black.

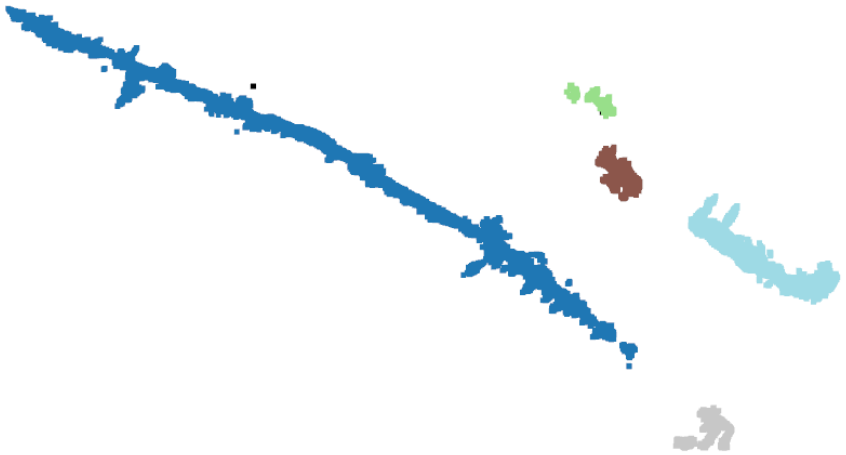

Figure 4.28: Clustering buildings of cloud 8088 with eps value 1.75. Noise points are colored in black

Figure 4.29: Clustering cars of cloud 741 with eps value 0.5. Noise points are colored in black.

Figure 4.30: Clustering cars of cloud 741 with eps value 1.25. Noise points are colored in black

Figure 4.31: Clustering cars of cloud 741 with eps value 1.75. Noise points are colored in black

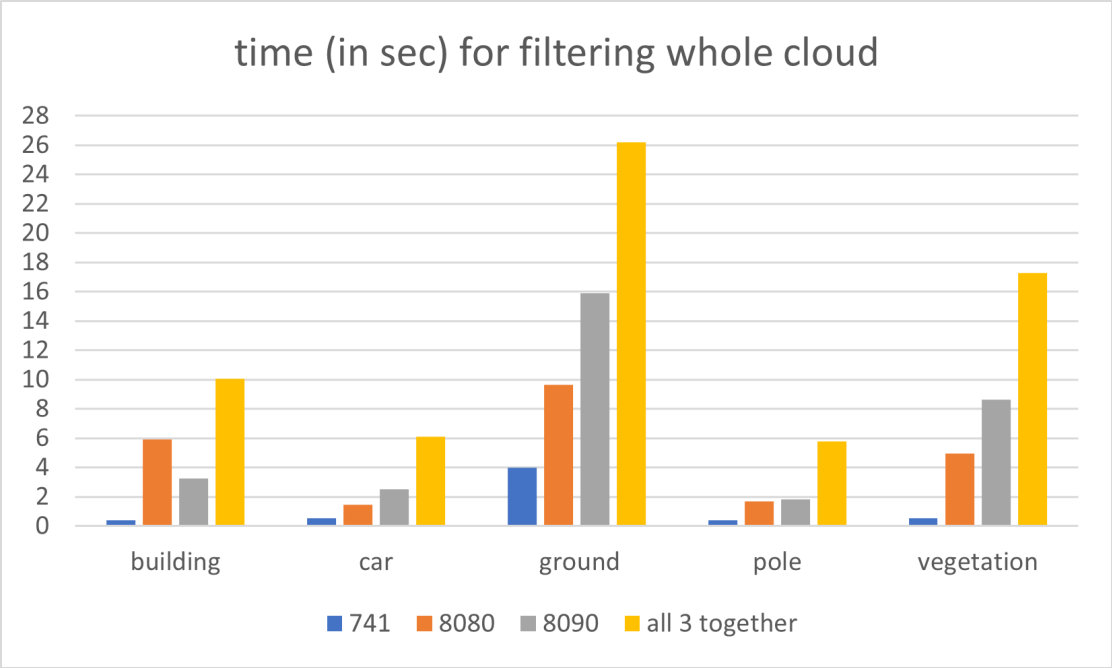

Figure 4.32: The run-time for filtering a class performed on multiple clouds as well as all of them together in one scene

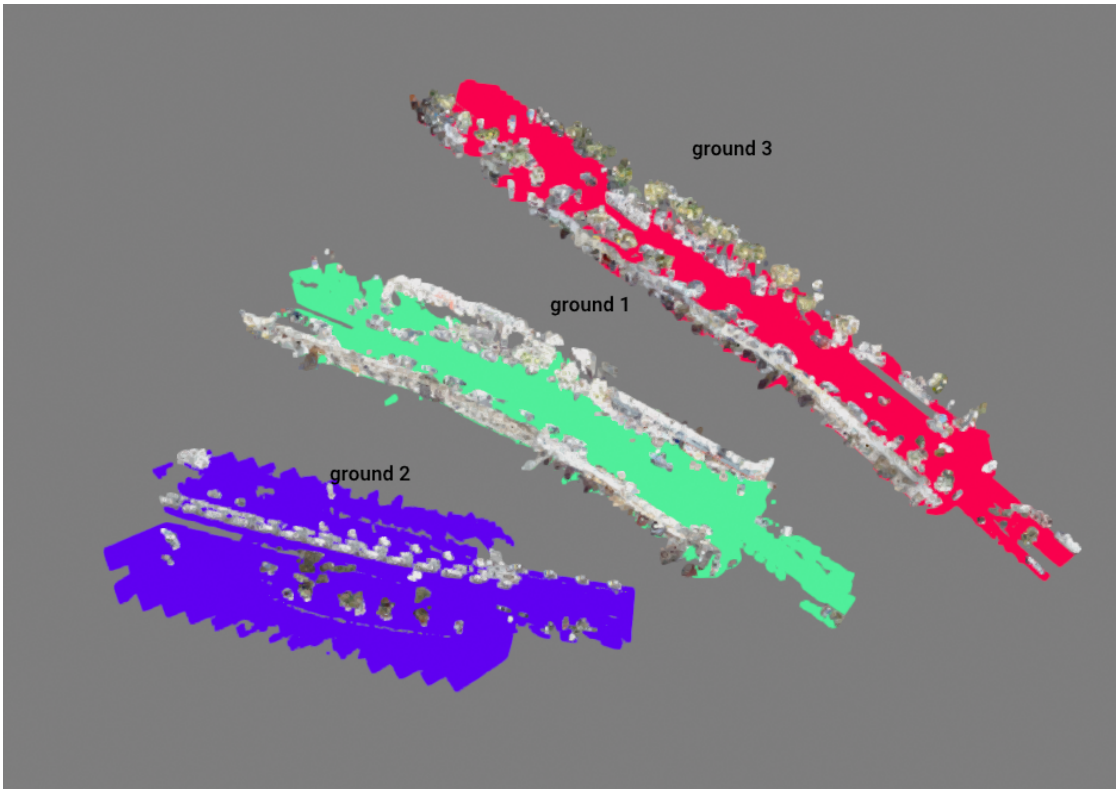

Figure 4.33: Filter ground when clouds 741, 8088, and 8090 are in a scene

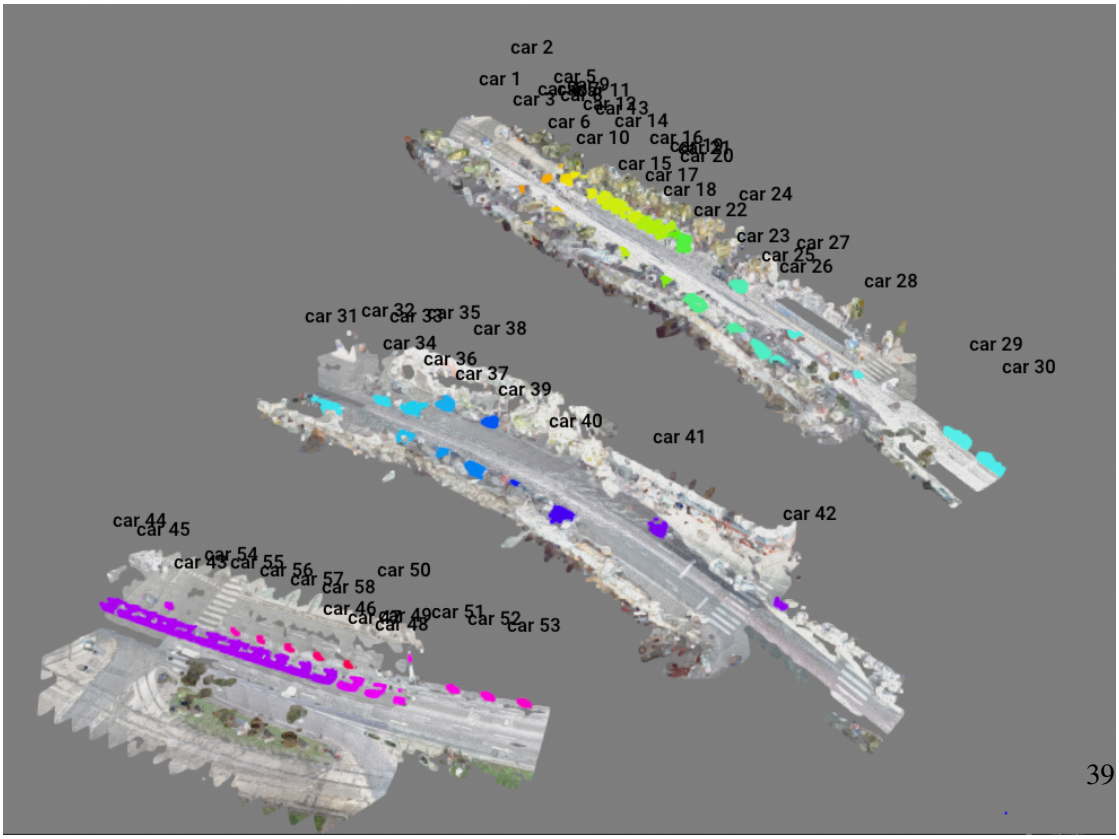

Figure 4.34: Filter cars when clouds 741, 8088, and 8090 are in a scene

CHAPTER 5

Future Work

5.1 DBScan Variation

As described in the 'Evaluation' chapter currently there is no way to figure out good values for 'eps' other than trying different ones out. Therefore one aspect that can be improved in the clustering process is to automatically calculate a good 'eps' value so that users do not have to spend time trying different ones out. There are a couple of variations of DBScan available that can do that, e.g. this one [7]. Using one of those should therefore improve that aspect of the clustering process.

5.2 Multiple Filtering & Object Selection

Currently, it is only possible to filter clouds with a single class. Furthermore, it is only possible to select one object at a time. Therefore the software can be improved by adding the option to filter clouds with a combination of different classes which should end up saving some time since multiple classes can be processed at the same time. Besides adding the option to select multiple objects at the same time can be quite helpful, especially for fixing classification mistakes. If one part of a building is classified as 'vegetation' and the other part is classified as 'building' it would be quite helpful to be able to select both and merge those into the desired class. Not only fixing mistakes would be easier, but being able to process multiple objects at the same time should also end up saving some time.

5.3 Classification algorithm

As mentioned numerous times it currently is not possible to classify humans with the GRanD-Net algorithm when using the pretrained model of the SHREC dataset. In order to reduce the misclassification of humans in clouds, another algorithm that has a pretrained model with humans could be used. If none are available a model of our dataset could be trained with the

GRanD-Net algorithm and then be used to classify the clouds which should improve the results. Using the RandLA-Net algorithm [17], which GRanD-Net is inspired by, and the SemanticKITTI dataset [14] [16] would be an option. That dataset contains human beings, cyclists, and the same street classes as the SHREC dataset [1].

5.4 Laz files

Currently, users can only save the classification and changes of clouds in a laz file. In order to improve usability different options such as creating an external txt file with the classification and cluster values could be created. By doing that users are not forced to save their files in the laz format.

Bibliography

[1] https://drive.google.com/drive/folders/1wiedl8lskkcq7elxgfypv54e71tdjwe5. Accessed: 2022-08-10.

[2] https://github.com/isl-org/open3d/blob/master/cpp/open3d/geometry/geometry3d.cpp. Accessed: 2022-08-10.

[3] https://laspy.readthedocs.io/en/latest/intro.html. Accessed: 2022-08-10.

[4] https://laspy.readthedocs.io/en/latest/lessbasic.html#extra-dimensions. Accessed: 2022-08-10.

[5] https://manifold.net/doc/mfd9/las,_laz_lidar.htm. Accessed: 2022-08-10.

[6] https://www.dgpf.de/src/tagung/jt2019/proceedings/proceedings/papers/12_3lt2019_falkner_eysn.pdf. Accessed: 2022-08-23.

[7] https://www.researchgate.net/publication/330248426_an_improved_dbscan_method_for_lidar_data_segmentation_with_automatic_eps_estimation. Accessed: 2022-08-10.

[8] http://www.open3d.org/docs/release/getting_started.html. Accessed: 2022-08-10.

[9] http://www.open3d.org/docs/release/python_api/open3d.geometry.orientedboundingbox.html. Accessed: 2022-08-10.

[10] http://www.open3d.org/docs/release/python_api/open3d.geometry.pointcloud.html. Accessed: 2022-08-10.

[11] http://www.open3d.org/docs/release/python_api/open3d.visualization.gui.html. Accessed: 2022-08-10.

[12] http://www.open3d.org/docs/release/python_api/open3d.visualization.gui.scenewidget.html. Accessed: 2022-08-10.

[13] Kiran Akadas and Shankar Gangisetty. *3D Semantic Segmentation for Large-Scale Scene Understanding*, pages 87–102. 02 2021.

[14] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[16] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[17] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, 2020.

[18] Manuel Keilman. Immersive Redesign, Bachelor thesis, Institute of Visual Computing & Human-Centered Technology, Vienna University of Technology, 2022.

[19] Kiran Akadas. https://github.com/kiranakadas/grandnet. Accessed: 2022-08-10.

[20] Tao Ku, Remco C. Veltkamp, Bas Boom, David Duque-Arias, Santiago Velasco-Forero, Jean-Emmanuel Deschaud, Francois Goulette, Beatriz Marcotegui, Sebastián Ortega, Agustín Trujillo, José Pablo Suárez, José Miguel Santana, Cristian Ramírez, Kiran Akadas, and Shankar Gangisetty. Shrec 2020: 3d point cloud semantic segmentation for street scenes. *Computers & Graphics*, 93:13–24, 2020.

[21] Shi Na, Liu Xumin, and Guan Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 63–67, 2010.

[22] C. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[23] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[25] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.