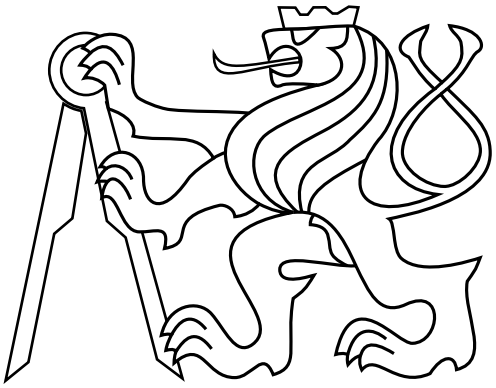
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics



Improving Detection by Exploiting Dynamics in the Lidar Data

Bachelor's Thesis

Vojtěch Bartek

Prague, May 2022

Study programme: Open Informatics, Bachelor
Branch of study: Artificial Intelligence and Computer Science

Supervisor: Ing. Patrik Vacek

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bartek Vojtěch** Personal ID number: **495543**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Improving Detection by Exploiting Dynamics in the Lidar Data

Bachelor’s thesis title in Czech:

Vylepšení detekcí pomocí modelování dynamiky v lidarových datech

Guidelines:

The task of detecting dynamic objects in the autonomous driving scenarios are crucial for vehicle navigation. The moving objects can change their movement and became dangerous to ego vehicle. These objects in traffic scenarios are constrained by class-specific dynamics and geometrical features. In the thesis, we will focus on exploiting such information to provide additional supervision from unlabelled data.
1) Research current unsupervised and self-supervised methods applied to lidar point clouds for detection of objects in autonomous driving scenarios.
2) Apply existing methods or design your own for separation of dynamic and static objects in lidar scenes. Focus on the usual scenario, where the amount of unlabelled data are plentiful. Try to leverage physically consistent and explainable features.
3) Focus on dynamics and sequentialness of the point clouds to create annotations for the foreground classes. The proposed method will provide additional labels from unlabelled dataset as the Teacher for the student model meant for online inference. The newly created annotations will be used to train 3D perception task.
4) Measure the extra benefit of proposed annotation method against already estabilished architectures.

Bibliography / sources:

[1] Z. Gojcic, O. Litany, A. Wieser, L. J. Guibas, and T. Birdal, “Weakly Supervised Learning of Rigid 3D Scene Flow,” in Proc. of the IEEE Conf. On Computer Vision and Pattern Recognition (CVPR), 2021.
[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss,and J. Gall, “SemanticKITTI: A Dataset for Semantic Scene Understandingof LiDAR Sequences,” inProc. of the IEEE/CVF International Conf. onComputer Vision (ICCV), 2019.
[3] I. Bogoslavskyi and C. Stachniss, “Fast range image-based segmentation ofsparse 3d laser scans for online operation,” in2016 IEEE/RSJ InternationalConference on Intelligent Robots and Systems (IROS), pp. 163–169, 2016.

Name and workplace of bachelor’s thesis supervisor:

Ing. Patrik Vacek Vision for Robotics and Autonomous Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **24.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Patrik Vacek	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgments

I promise, I'm honored, I'd like to thank God, my mamma and father
I'd like to thank Tricia, the mother of my daughter
I couldn't have done it without you all in my corner
Especially the fans, been here since the beginning
Supported the music, allowed us to be independent
— Macklemore & Ryan Lewis, Light Tunnels (feat. Mike Slap)

I want to express my utmost gratitude to Ing. Patrik Vacek for being a great supervisor, mentor, and friend. Thank you for your guidance and patronage.

I would like to thank my family for supporting me during my studies and allowing me to focus on the university entirely. Last but not least, I want to acknowledge my friends, who motivated me, helped me and made me laugh during the last three years.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 2022

Abstract

Detecting dynamic objects from a 3D Point Cloud (PCL) is crucial for many modern problems, such as collision detection for self-driving cars or mobile robotics. Currently, this field is dominated by supervised methods. However, they require a lot of labeled data for training, and labeling a PCL is very time and resource-consuming. This thesis proposes an offline unsupervised approach for dynamic object detection, focusing on high precision, that can generate labels. It then explores how supervised models can benefit from those automatically-generated labels.

We evaluate Principal Component Analysis (PCA), Random Sample Consensus (RANSAC), range image-based method, and pillars method for ground removal, which is crucial for good spatial separation of objects resulting in better segmentation and object detection.

Lastly, we propose a new semi-supervised method. We perform segmentation on two consecutive frames, find point mapping between the two frames and enforce the same classification for the corresponding points.

The results of a model trained on automatically-generated labels together with mixed fraction of ground truth surpass the fully learned detector with fewer annotations needed.

Keywords Point cloud, Object detection, Supervised model

Abstrakt

Detekce dynamických objektů z 3D mračna bodů (PCL) je zásadní pro mnoho moderních problémů, jako jsou například detekce kolize pro samořiditelná auta nebo pohybující se roboty. V dnešní době dominují v těchto úlohách supervizované modely. Tyto modely vyžadují značné množství anotovaných dat a anotace lidarového snímku je velmi časově náročná. Tato práce navrhuje nesupervizovaný přístup neběžící v reálném čase pro detekci dynamických objektů, který se zaměřuje na vysokou preciznost a který je schopen generovat anotace. Dále tato práce zkoumá využití těchto vygenerovaných anotací při trénování supervizovaného modelu.

Vyhodnocujeme Principal Component Analysis (PCA), Random Sample Consensus (RANSAC), metodu založenou na vzdálenostech a metodu Sloupců pro odstranění země. Odstranění země je důležitý krok pro správné prostorové oddělení objektů, což vede k lepší segmentaci a detekci objektů.

V neposlední řadě navrhujeme novou semi-supervizovanou metodu. Provedeme segmentaci na dvou po sobě jdoucích snímcích, určíme korespondence mezi těmito snímky a vynutíme stejnou klasifikaci pro odpovídající body.

Výsledky modelu natrénovaném na kombinaci našich automaticky vygenerovaných a menší části originálních anotacích překonávají model plně naučený na ručně vytvořeních anotacích.

Klíčová slova Mračna bodů, Detekce objektů, supervizovaný model

Abbreviations

PCL Point Cloud

GT Ground Truth

PCA Principal Component Analysis

RANSAC Random Sample Consensus

BFS Breadth-First Search

CNN Convolutional Neural Network

IoU Intersect over Union

LiDAR Light Detection and Ranging

DBSCAN Density-based spatial clustering of applications with noise

KNN K-Nearest Neighbors

MOS Motion Object Segmentation

Contents

Chapter 1

Introduction

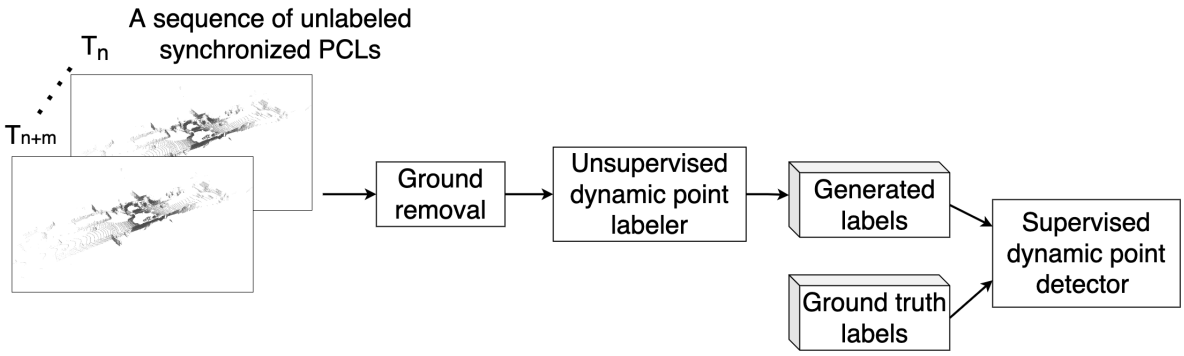


Figure 1.1: Overview of the main pipeline of this thesis. First, we synchronize a sequence of PCLs and remove the ground points. Then all points are labeled using our unsupervised method. We prepare a new mixed data set for a supervised model to train from and evaluate the model's performance on original ground truth labels.

Computer vision is a field of computer science that focuses on exploring and enabling computers to perceive and process images and videos measured from sensors such as the camera. Whether in robotics or autonomous driving, computers need to understand the surrounding scene to safely and correctly interact with the environment. Detecting dynamic objects is necessary to predict and avoid collisions or track objects. In recent years, more and more scientists and engineers have been exploring the capabilities of a Light Detection and Ranging (LiDAR) as a sensor for object detection because of its ability to measure precise distances for each point it reaches [1, 2].

State-of-the-art methods for dynamic object detection from PCL are based on supervised or semi-supervised approaches, creating a great need for manually labeled data [3, 4, 5]. This work proposes a method to detect dynamic objects without the need for labels at the cost of greater computational power.

We focus on high precision (defined in Chapter 3), as it would not be beneficial for a model to be trained on wrongly labeled data. High precision means that almost everything our method labels as dynamic is truly dynamic. Consequently, the method's recall is lower - meaning that it does not label all dynamic objects. Lower recall does not pose a significant issue (as long as it is not close to zero) since we can alter the loss function during the training of a neural network in such a way that we do not penalize classification on the data for which we have no labels. We used the SemanticKITTI data set [6, 7], which provides labels for each point in a PCL with various sequences recorded primarily in urban areas.

We explore three different approaches to solving this problem and use our best method

to generate labels that we use to train a Convolutional Neural Network (CNN) and evaluate the impact of our annotations. This idea is depicted in Figure 1.1.

To decide on the motion of an object, we need to observe it in a short sequence. Had the LiDAR sensor remained motionless during the whole recording process, we could concatenate arrays of points and plot all frames at once to see the dynamics of the scene. This is not the case, as the LiDAR sensor is mounted on the roof of a car, which rarely remains motionless. To compensate for the sensor's movement, we first have to synchronize the PCLs. This process is explained in Chapter 2.

Because of how our methods work, we need to remove the ground, as it enables us to perform clustering and speeds up the following processes. We know that we will not lose any dynamic object by removing the ground from our prior knowledge of the situation. In Chapter 3 we explore multiple ground removal algorithms and compare them.

Lastly, in Chapter 8, we explore a self-supervised loss function that could be used to improve an already trained model further using correspondences between two consecutive frames.

1.1 Related work

Related LiDAR-based object detection work includes [5], which uses multiple LiDAR sensors and bounding box fitting to estimate the object's centroid, which is later used for tracking. Kiran et al. [8] propose using a 3D prior map to obtain dynamic objects by subtracting background. This approach, of course, requires a 3D prior map and does not work in a previously unmapped location. Bogoslavskyi and Stachniss [9] perform range image-based segmentation with an angle-based approach for ground removal, which we have evaluated in Chapter 3. Another work [10] designed a fast fully convolutional neural network, which can predict the road from the bird's eye view projection of the scene. However, this approach does not predict the road behind obstacles, e.g., vehicles, and therefore makes mistakes in these points. Approaches [11, 12] can separate ground from non-ground points, which can be even improved by the usage of The Jump-Convolution-Process [13].

PointRCNN [14] is a single-frame neural network that operates on raw PCL using encoders to improve spatial and semantic features. Yin et al. [15] work with point cloud video and Spatio-temporal feature extraction to detect dynamic objects which may be temporarily occluded. In [16], the authors propose an unsupervised, iterative method for object detection that combines the LiDAR PCL and 2D images to create annotations for supervised models. VoxelNet [17] uses voxel-based feature extraction and convolution for object detection. In [18], the authors use two self-supervised losses for scene flow estimation, and we have also explored this approach in Chapter 8. Chen et al.[3] propose a CNN LMNet that uses residual range images; we use this model for evaluation of our data in Chapter 7. We distill our newly created labels via pseudo-labelling framework [19].

In general, neural network-based methods provide accurate online results at the cost of the loss of explainability and manual labor required to create the annotation. Our approach retains explainability and could lessen the need for labeled data in training semantic segmentation networks [3, 20, 21].

Chapter 2

PCL synchronization

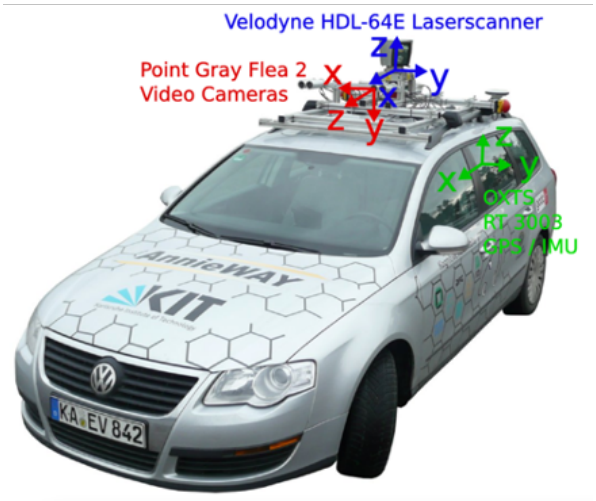


Figure 2.1: Schematic of the setup used for recording the KITTI [7] data set, taken from their website.

PCL is recorded by a LiDAR sensor, described in Figure 2.2, which rotates at a certain frequency. As a result, if the sensor is moving (e.g. mounted on the roof of a moving car, as shown in Figure 2.1), even a static point $\{x_t, y_t\}$ at time t has different coordinates $\{x_{t+1}, y_{t+1}\}$ at time $t + 1$, as depicted in the Figure 2.3.

Since we know in what direction the sensor moved (from the Inertial measurement unit - IMU), this issue is solved simply by synchronizing (transforming) each PCL frame into the same reference coordinate system with a transformation matrix $\mathbf{M} \in \mathbb{R}^{4x4}$, created from calibrations and time poses.

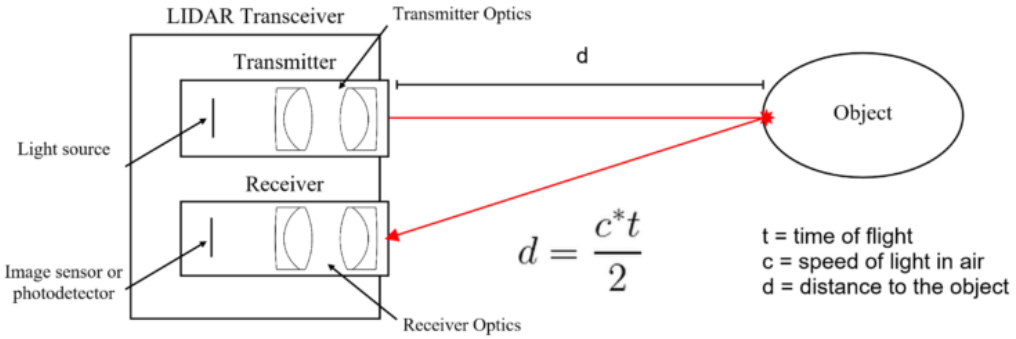


Figure 2.2: Schematic for LiDAR sensor taken from [22]. The distance is calculated from the time of flight and the speed of the light ray. From the angle of incidence, we can calculate the position of the object the ray has ricocheted off.

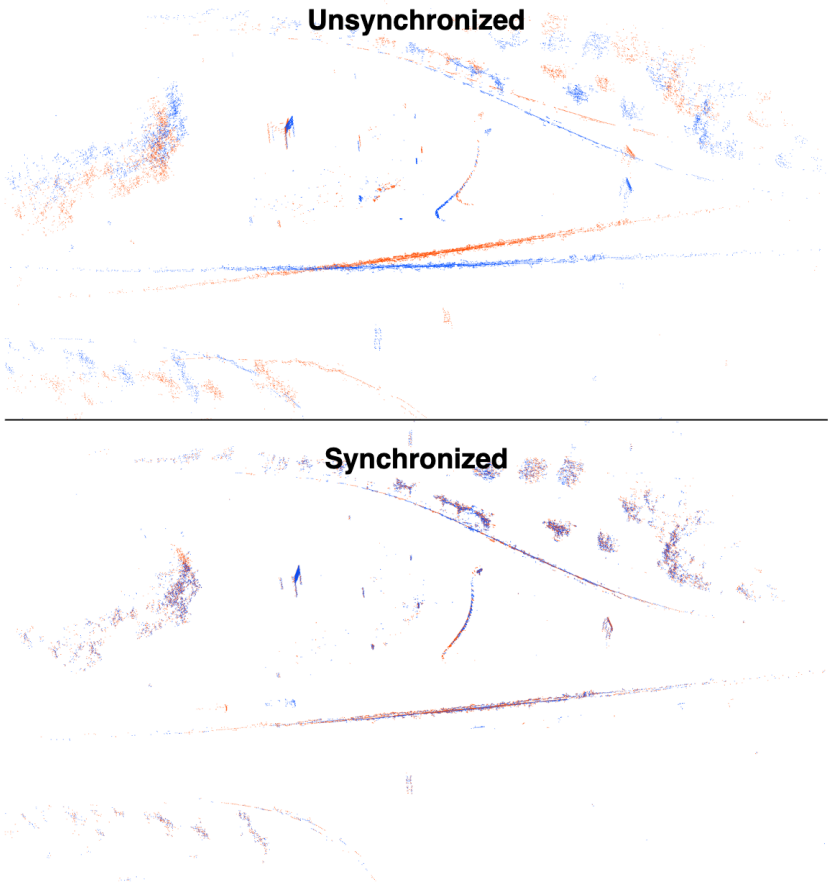


Figure 2.3: Top - two unsynchronized PCLs; Bottom - the same two PCLs but synchronized.

The matrix **M** is obtained as shown in equation 2.1. The first three rows of the matrix **A** are the calibrations of the mounted sensor, and the last row is the vector [0 0 0 1]. **B** is a matrix of poses, which we obtain for each time frame, representing the movement of the sensor between frames.

$$\mathbf{M} = \mathbf{A}^{-1} \times (\mathbf{B} \times \mathbf{A}) \tag{2.1}$$

where the symbol $\times$ denotes matrix multiplication. To synchronize a PCL frame denoted $\mathbf{P} \in \mathbb{R}^{Nx3}$ made up of vectors of points, we add a column of ones, and transform it as follows:

$$\mathbf{P}_{synchronized} = (\mathbf{M} \times \mathbf{P}^T)^T \tag{2.2}$$

and the first three rows of this transformed matrix $\mathbf{P}_{synchronized}$ are the coordinates XYZ of the synchronized points.

Chapter 3

Ground removal

After synchronizing a PCL sequences, we want to remove the ground since it is not a dynamic object. This will significantly reduce the number of points to search from, and it is essential for a good segmentation. In this section, we compare a few algorithms we have tried for ground removal on a single PCL frame. The comparison metrics used for evaluation are as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$IoU = \frac{TP}{TP + FP + FN} \tag{3.3}$$

where TP is the number of true positives, FP is the number of false positives and FN is the number of false negatives. Intersect over Union (IoU) is a standard evaluation metric for segmentation. One can see ground removal as a binary segmentation task.

All the following methods are implemented in python programming language [23] using the NumPy library [24]. We might further speed up the methods using a lower-level language, such as C.

We know which points belong to the ground thanks to the SemanticKITTI labels. With this information, we can evaluate the discussed algorithms. A PCL with the ground and the same PCL without ground can be seen in Figure 3.1.

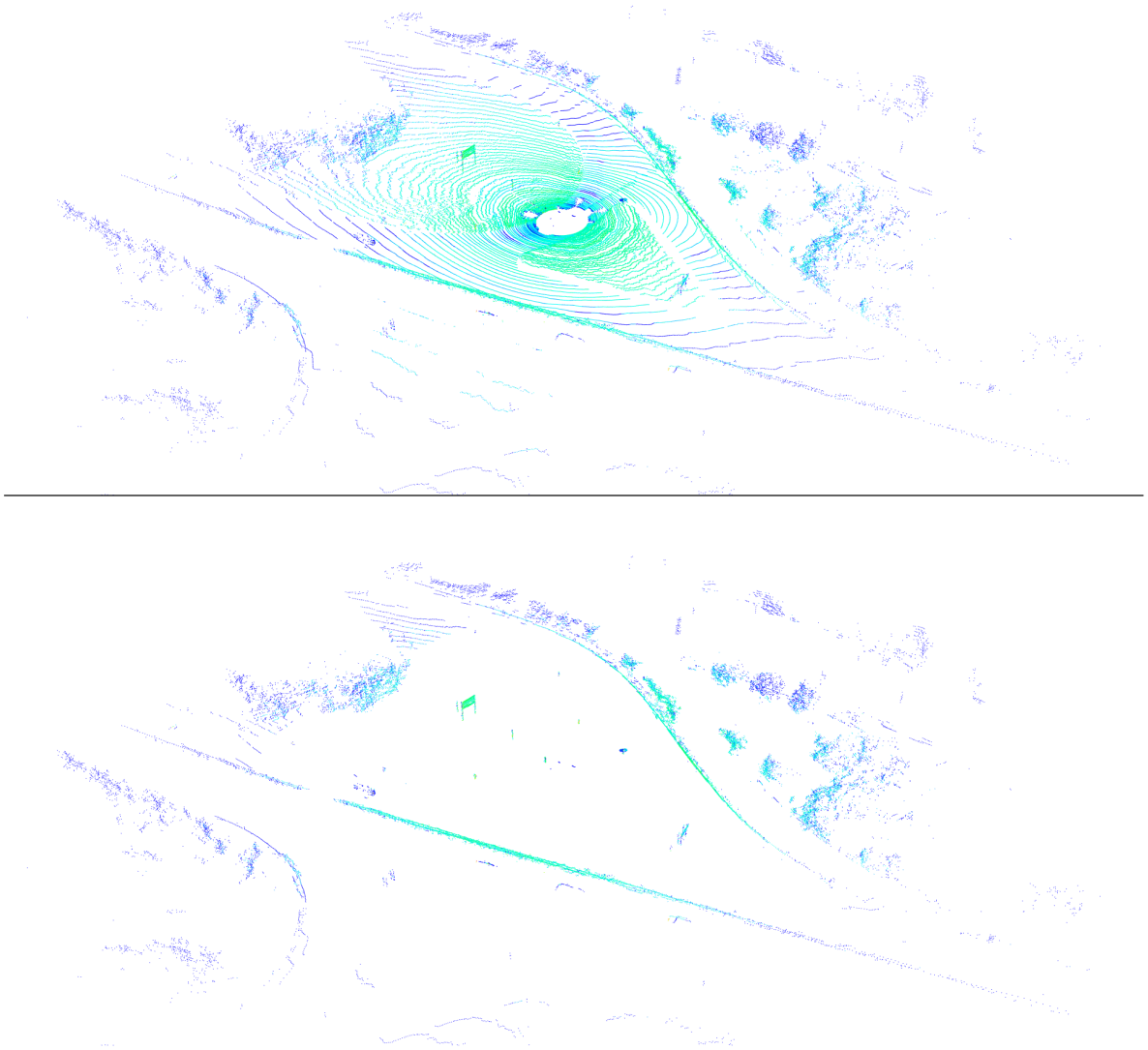


Figure 3.1: Top - single PCL frame with ground; Bottom - the same PCL with ground removed using labels.

3.1 PCA

We compute the eigenvalues and the eigenvectors of a matrix $\mathbf{M}^T \times \mathbf{M}$, where $\mathbf{M} \in \mathbb{R}^{Nx3}$ is a matrix of 3D points from PCL. We chose the eigenvector with the lowest eigenvalue as the normal vector of the plane, which we will declare as ground and categorize all points closer than ϵ to the plane as ground. In our implementation, we have selected ϵ as 0.2 - 20 centimeters.

Observing the Figure 3.2, this method fails to remove the ground that is further away from the origin (LiDAR position) and it also removes parts of trees and other non-ground objects. Therefore, it is not suitable for our needs.

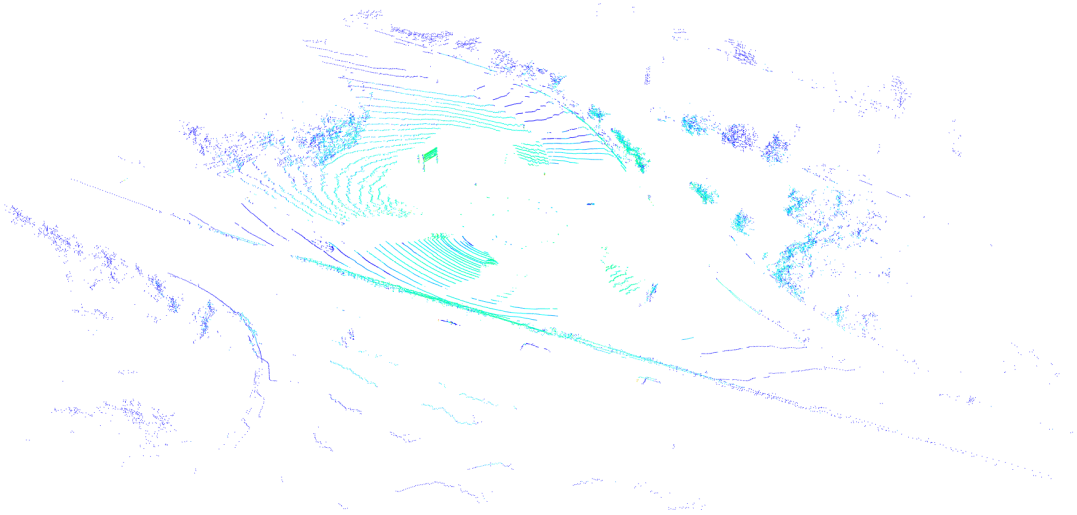


Figure 3.2: PCL after removing the ground with PCA.

3.2 RANSAC

We randomly choose 3 points, create a plane containing those points, and measure the number of points closer to this plane than ϵ. We repeat the process for a γ number of iterations, select the plane that fits the most points, and declare all the points closer than ϵ to the plane as a ground. In our experiments, ϵ is set to 0.2 - 20 centimeters, and γ is set to 6.

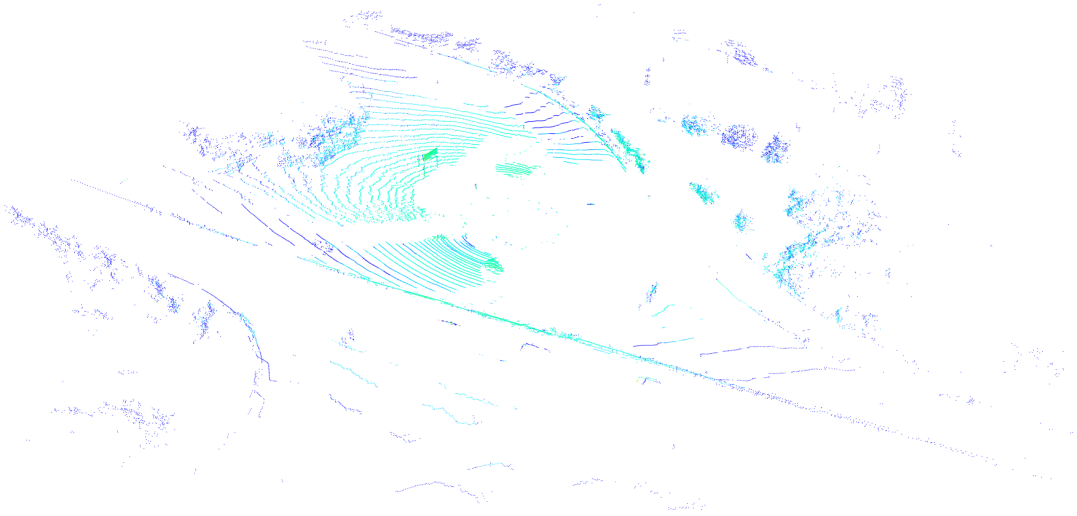


Figure 3.3: PCL after removing the ground with RANSAC.

Ideally, we would need to run more iterations than 6, but the time complexity of this method is very high. As with PCA, it tries to fit a plane to the ground, but in real world, ground is rarely a perfectly flat surface. This method still fails to meet our needs.

3.3 Range image ground removal

Inspired by [9], we project the point cloud onto a cylindrical image and compute the Euclidean distance per pixel to obtain a range image. We then try to repair the range image by filling empty projection points with the average of the previous and the next row, depicted in Figure 3.5, and apply the Savitsky-Golay smoothing [25].

Our LiDAR has 64 beams, therefore we have 64 rows in our range image. We compute the α angles between each adjacent pair of rows as follows:

$$\begin{aligned} \alpha &= \arctan(\Delta z, \Delta x) \\ \Delta z &= |R_{r-1,c} * sin\xi_a - R_{r,c} * sin\xi_b| \\ \Delta x &= |R_{r-1,c} * cos\xi_a - R_{r,c} * cos\xi_b| \end{aligned} \tag{3.4}$$

with $R_{r,c}$ being the range in row r and column c and ξ_a, ξ_b are vertical angles of the laser beams corresponding to rows $r-1$ and r. We obtain an angle matrix $\mathbf{M}_\alpha = [\alpha^r_{r-1,c}]$ where r and c are row and column coordinates of the range readings from the range image. An example explaining this idea can be seen in Figure 3.4.

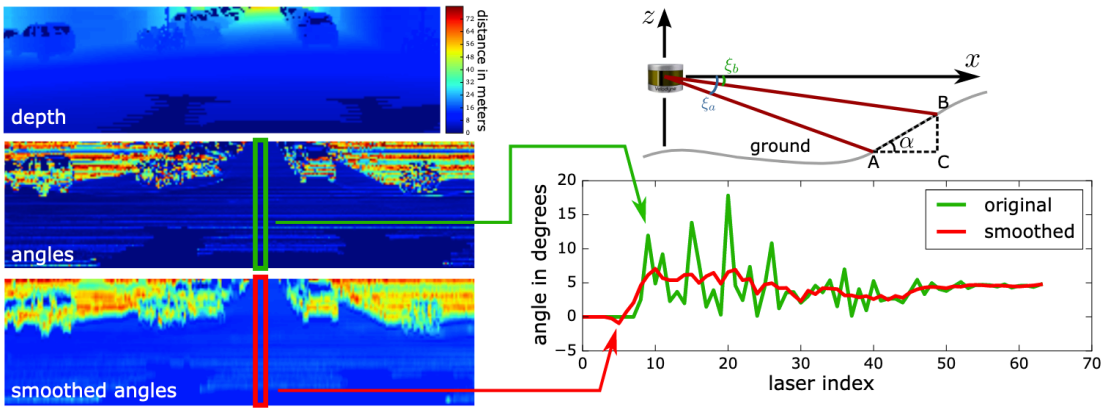


Figure 3.4: Schematics of range image method, taken from [9].

We then label all points in the first row that have angles smaller than 45° as ground and start a Breadth-First Search (BFS) from those points. If $|\mathbf{M}_{(r,c)} - \mathbf{M}_{(r_n,c_n)}| < 5°$, where r_n and c_n are the neighbor coordinates, we label the neighbor point as a ground and add it to the queue. The final result can be seen in Figure 3.6.

This method approaches the problem with a better idea than the previous methods. It does not suppose the ground is flat but allows slight gradual differences. However, it still leaves us with many ground points that would later interfere with the following steps.

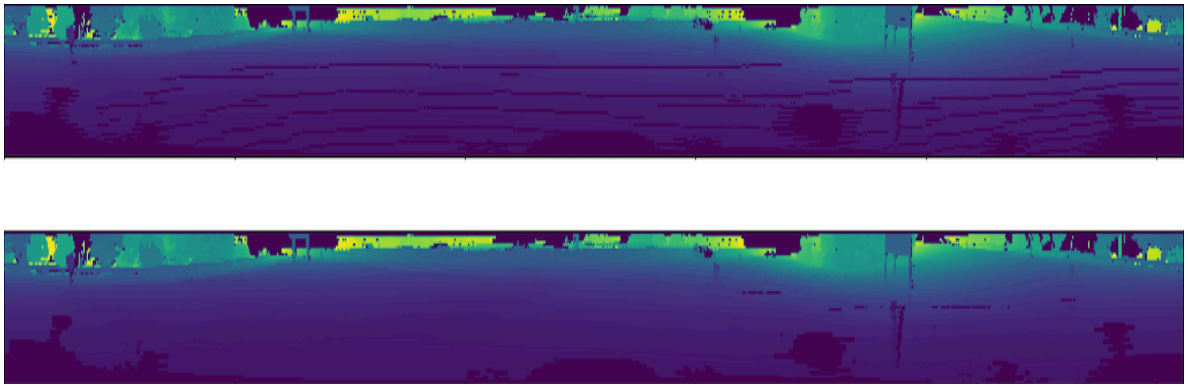


Figure 3.5: Top - unrepaired range image; Bottom - repaired range image.

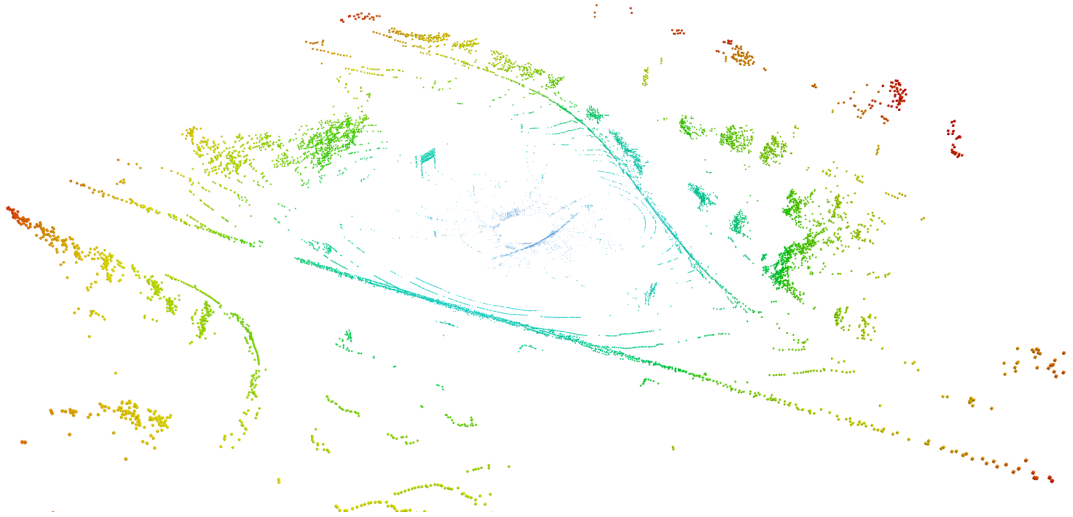


Figure 3.6: PCL after removing ground with range image method, the coloring of points is different because of the loss of point intensities during projection.

3.4 Pillars

So far, all the previous methods have left us with too many remaining ground points. And so, we have decided to design our own algorithm. One that would better meet our needs to help us separate dynamic objects from static ones.

Our idea is to divide the space into thin pillars and label the lowest points inside each pillar as ground. This idea is similar to voxelization, introduced in Chapter 4. Sometimes it labels some non-ground points as ground (e.g., part of a roof); however, this rarely happens, as most dynamic objects are often vertical. This fact is supported by this method's very high precision seen in Table 3.1.

We have used a 0.2 by 0.2 meters square base for the pillars and labeled as ground all the points whose height difference from the lowest point is less than 0.1 meters. The results can be seen in Figure 3.7. One can see significant difference between the result of this method compared to the others. This approach removed most of the ground points.

Table 3.1: Ground removal performance evaluation, measured in %.

Method	Precision	Recall	IoU	Time (sec)
PCA	61.13	34.83	30.29	0.50
RANSAC	80.42	58.21	52.55	29.51
Range image	75.94	59.18	51.76	<0.05
Pillars	**94.39**	**64.51**	**62.20**	<0.05

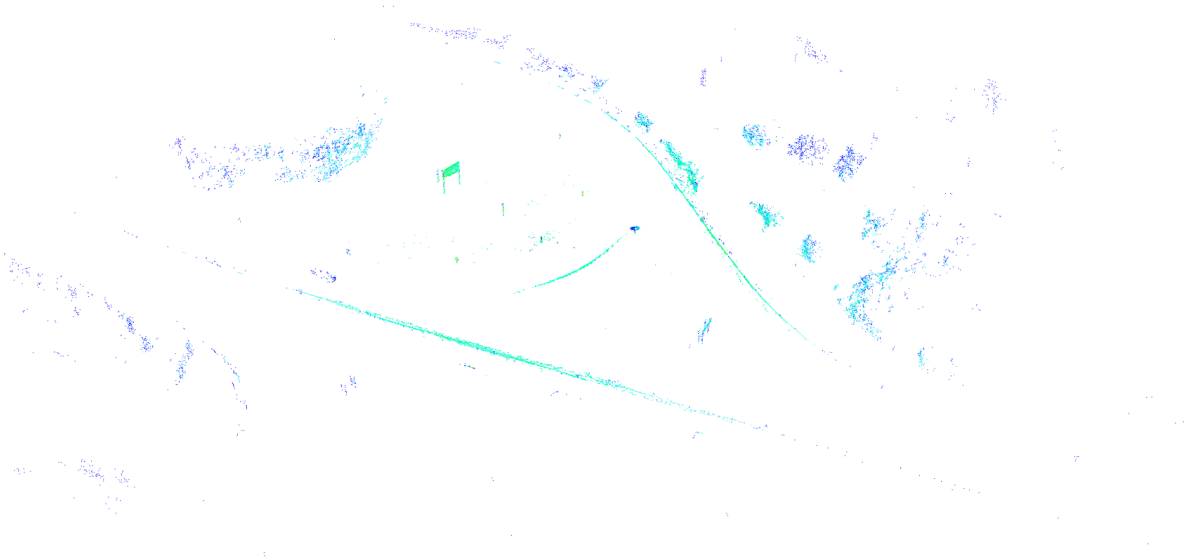


Figure 3.7: PCL after removing ground with pillars method.

The results in Table 3.1 are averaged across all 1101 PCLs in the SemanticKITTI sequence 1. We have selected this sequence as it has a wide variety of environments, and so it gives us a reasonable estimate of model performances. The PCA algorithm has all metrics worse than the others. RANSAC has high average precision but at a very long computation cost. The range image approach has a bit lower precision than RANSAC and performs similarly in terms of recall and IoU. Still, it runs substantially faster than RANSAC, and it can be observed that most unfiltered points are right around the ego. These could be filtered using PCA or RANSAC, which would now run much faster. The Pillars algorithm outperforms all of the other methods. Its high precision, recall, IoU, and fast execution time make it the perfect candidate for our task.

Chapter 4

Voxel-based approach

To decide whether a point belongs to a dynamic object, we need to observe some short sequence of frames. One approach could be to go over each frame of a sequence and find all points that have no neighbors around them in the previous or following PCL. If there are no neighbors from other times in some small proximity of the observed point, it is likely to be a point of a dynamic object. To help us decide what is close and to handle the problem with sensor imperfections, we split the observed space into n-dimensional cubes called voxels.

The main steps of this method are:

- create a voxel grid on multiple synchronized PCLs with removed ground
- measure the volume of each voxel based on time
- classify voxel, consequentially all points within this voxel, as dynamic if the said voxel had points only in one time
- perform density-based clustering to eliminate noise

4.1 Voxelization

We create a 2D voxel grid with a voxel size of 0.5 meter^2 and check for each voxel whether there are any points belonging to its area. Such a grid with only non-empty voxels visible can be seen in Figure 4.1. Let T be the number of time sequences across which we want to detect dynamic objects, we obtain a matrix $\mathbf{Q} \in \mathbb{N}^{TxXxY}$, where X is the grid size on the X axes and Y is the size on the Y axes.

The matrix at position $[t, x, y]$ contains one if there is at least one point in the voxel at time t at that position; otherwise, it contains zero. The voxel grid remains in the same position for all PCLs; we do not move the voxels in any way. This allows us to monitor how the content of voxels differs across time. Because of the synchronization, static points will remain at the same position (with some minor deviations that are negligible because of voxelization), so if $\mathbf{Q}[t, x, y] \neq \mathbf{Q}[t + 1, x, y]$, it is either because of a dynamic object moving into or out of the voxel, or because the LiDAR lasers have reached previously unreachable places as a result of the sensor moving.

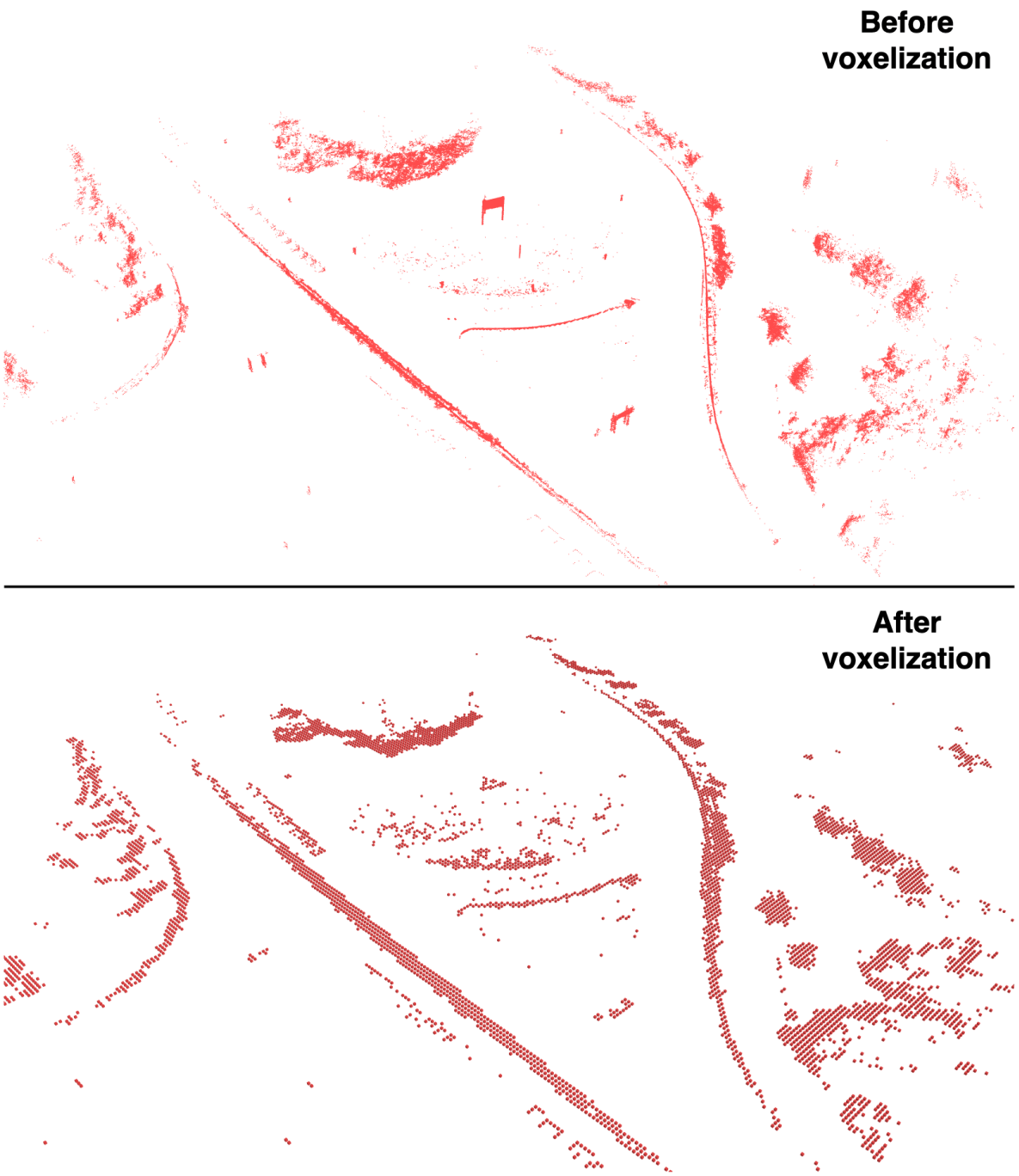


Figure 4.1: Top - 5 synchronized consecutive PCLs; Bottom - 2D voxelization of the same synchronized PCLs with voxel size of 0.5 meter^2, only showing non-empty voxels.

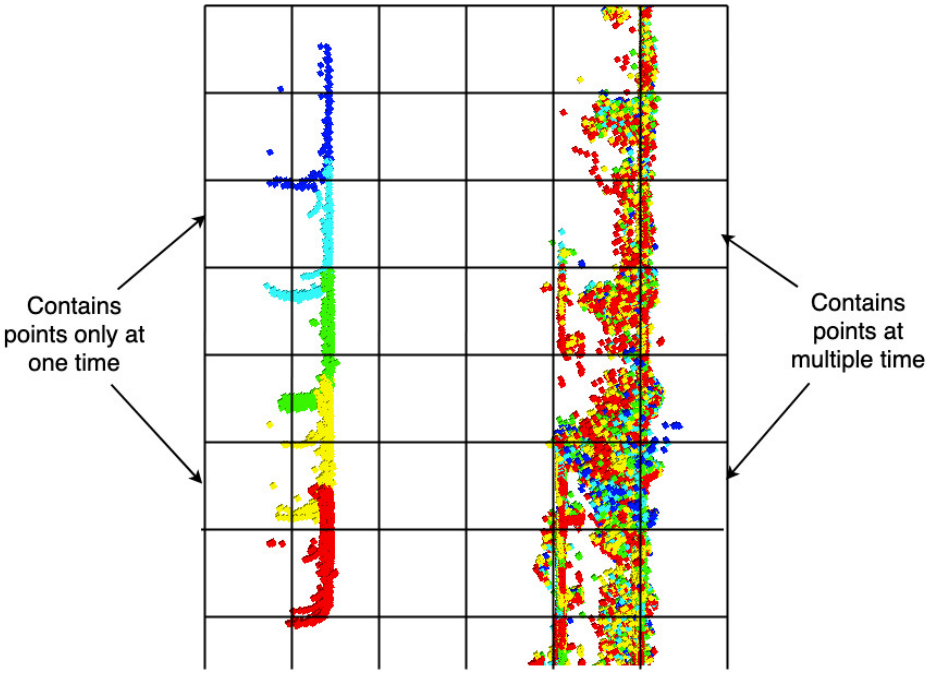


Figure 4.2: 2D voxel grid over 5 synchronized PCLs, each point is colored according to its time of recording. On the left is a moving car, and on the right is a road barrier. To simplify, our method selects voxels that contain points of only one color which means there are points present at only one time.

We sum **Q** across the time axis and select all voxels that have the sum equal to one, meaning that at one frame, there was any non-zero number of points, and at all other times, there were no points. Situation depicted in Figure 4.2. We discard all voxels that have the sum equal to one and belong to the first time sequence or the last, as many new points appear only in those sequences; therefore, they would be falsely classified as dynamic.

We do realize this criterion is rather simple and could be improved, but we have seen potential in another method presented in the next chapter.

We have used 2D voxelization as it has proven to be more efficient and robust than 3D voxelization. This approach works because of our prior knowledge that dynamic objects are all moving on the same plane, the ground.

4.2 Clustering

After filtering most static points with ground removal and voxelization, we are left with many sparse points and a few dense clusters of points. The sparse points represent imperfect measurement or random slight movement, for example, tree leaves. The dense clusters represent dynamic objects.

The dynamic points can be clustered together since the LiDAR sensor revolves at a frequency of 10 Hz, meaning that the dynamic objects do not travel long distances between each frame. Therefore, we can use clustering to filter out the noise. We have chosen the Density-based spatial clustering of applications with noise (DBSCAN) [26] as our clustering method.

DBSCAN requires two hyperparameters:

- ϵ - radius of a neighborhood for each point
- *min_samples* - minimum number of points in a neighborhood of an arbitrary point for this point to be considered a core point of a cluster

For the first clustering, we have set $\epsilon = 0.6$ (in meters); the reason behind this is that we have chosen voxel size of 0.5, meaning that if an object (e.g., a car) is divided between a few voxels, DBSCAN can still reach adjacent voxels and cluster the whole vehicle. We want to eliminate as much noise as possible; for this reason, we have to keep the ϵ parameter as small as possible. We have set the parameter *min_samples* to 8, which has proven by our observation to be a balanced choice that eliminates most static points while keeping most of the dynamic ones.

This still leaves us with some noise, but this can be filtered by removing all points outside the clusters and performing the clustering again with ϵ set higher to allow the clustering of dynamic objects across time frames, which consequently pushes *min_samples* higher. In our second clustering, we have set ϵ to 3 and *min_samples* to 30. The result can be seen in Figure 4.3.

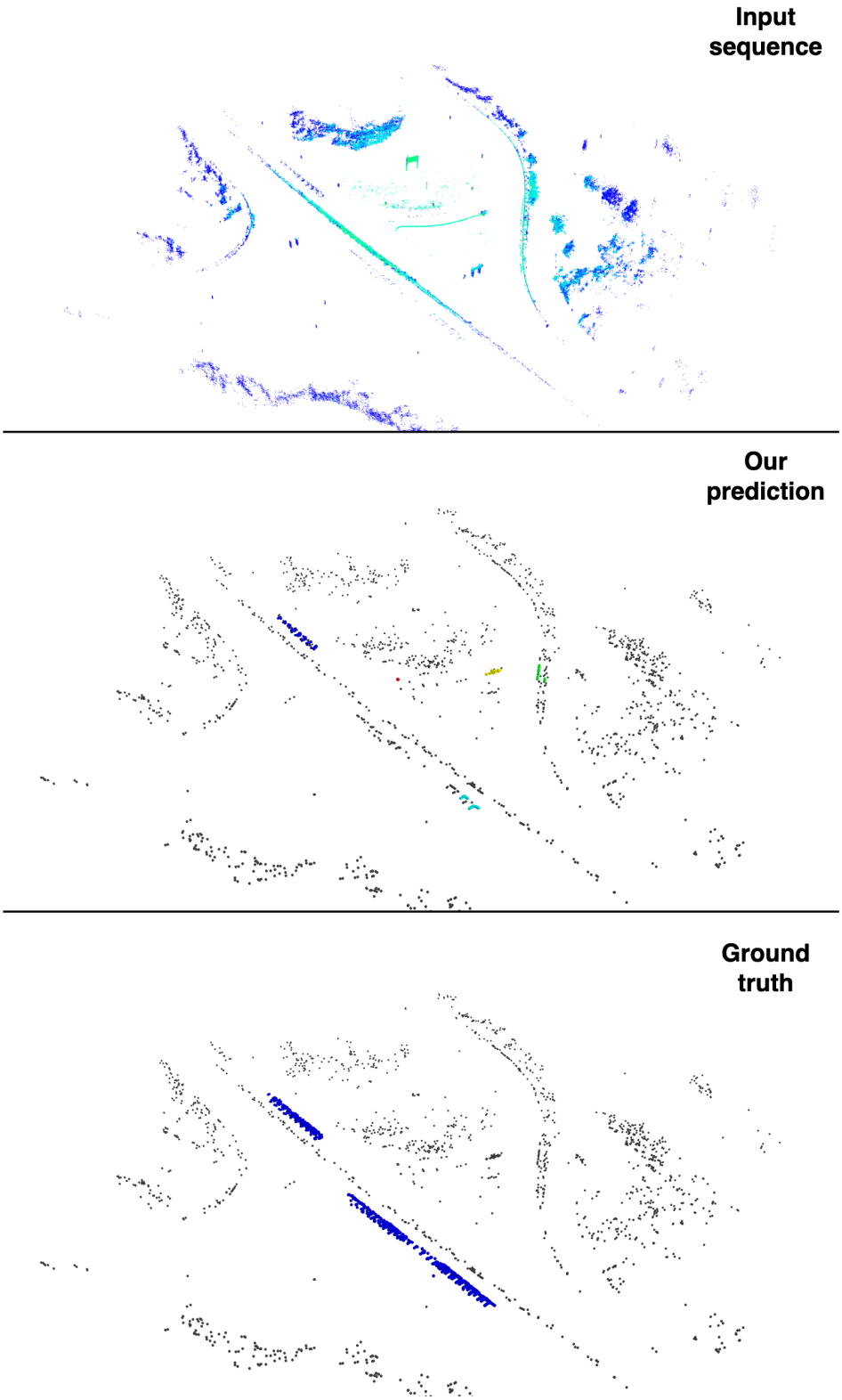


Figure 4.3: Top - input 10 synchronized PCLs; Middle - voxel method prediction after performing second clustering, note that the green, yellow, and red cluster are false positives; Bottom - ground truth dynamic points.

4.3 Evaluation

This method proved insufficiently precise for our needs and very time-consuming. Even on a very clear and simple example, this method failed to detect the whole car and instead falsely detected a large and static object as dynamic. It was clear, we would need a more sophisticated solution that would better incorporate the rules of motion and take into account the movement of the LiDAR sensor itself.

Chapter 5

Centroid-based approach

Instead of separating the space with voxels as in Chapter 4, we could generalize the objects we want to evaluate. One such way to approximate a cluster of points (an object) is by its centroid, the mean of all points of an object. By observing the movement of a centroid of a cluster over time, we can decide whether it is dynamic or static.

This method consists of the following steps:

- synchronize multiple PCLs with the ground removed
- perform segmentation by density-based clustering to obtain objects
- remove clusters of invalid sizes
- for each remaining cluster, compute the centroid of points for each timestamp
- observe the change of centroids over time

5.1 Pre-processing

To be able to work with whole objects rather than individual points, we cluster close points together with the DBSCAN algorithm described in Chapter 4.2. We perform clustering once on all synchronized PCLs (with removed ground) of some short sequence, and so we do not cluster each PCL individually. In this way, we get clusters that contain points from multiple timestamps, and we can observe changes over time within this cluster.

Let $T = \{T_1, ..., T_n\}$ be a set of synchronized PCLs. Had we carried out clustering on each PCL individually, we would have problems with correspondences - which cluster in time T_1 corresponds to some cluster from time T_2, and we could not observe the centroids across time. The result of clustering 10 PCLs is shown in Figure 5.1.

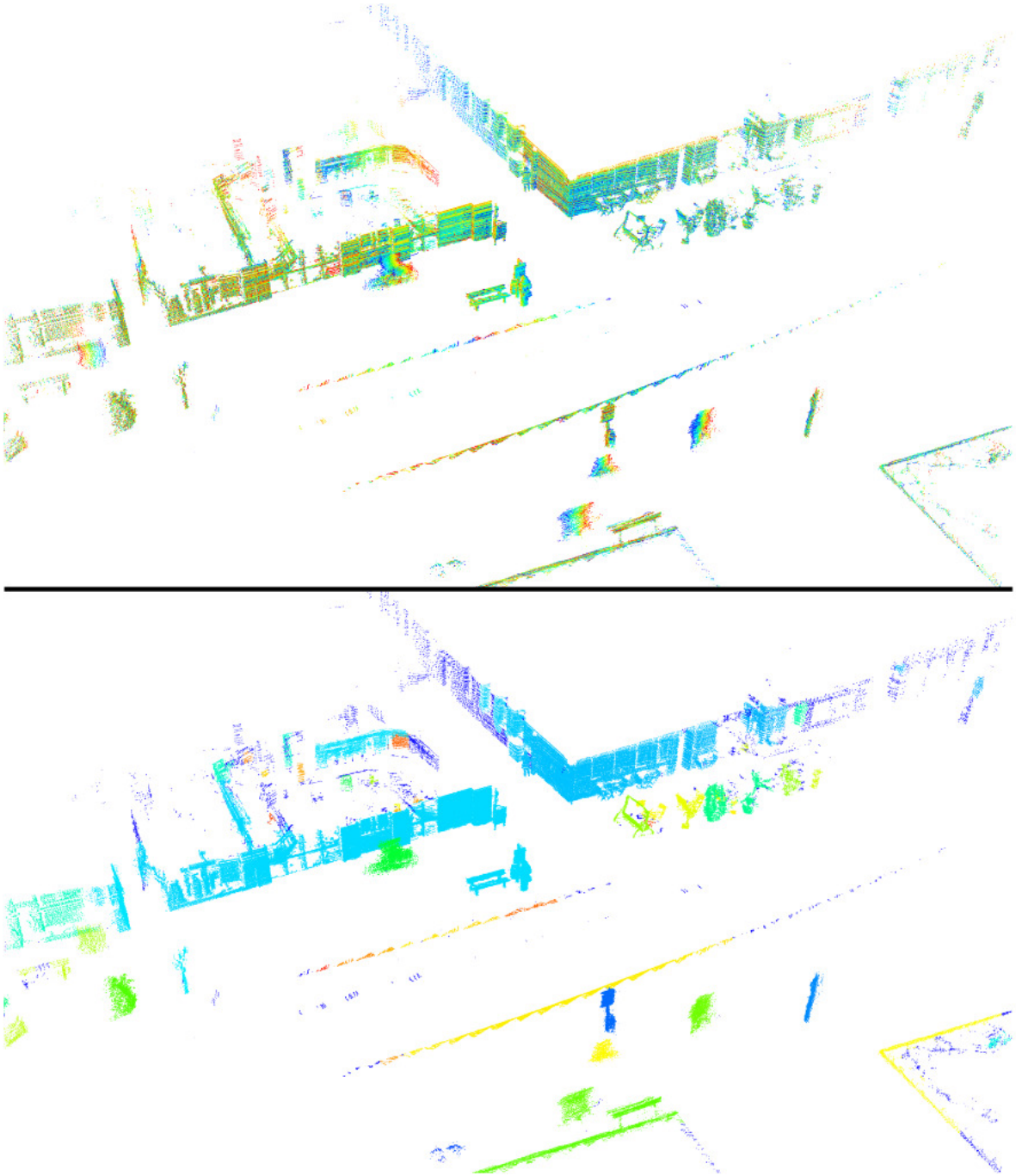


Figure 5.1: Top - 10 synchronized PCLs with removed ground; Bottom - the same 10 PCLs after clustering.

We can then remove all clusters that are too big and move on to the next step.

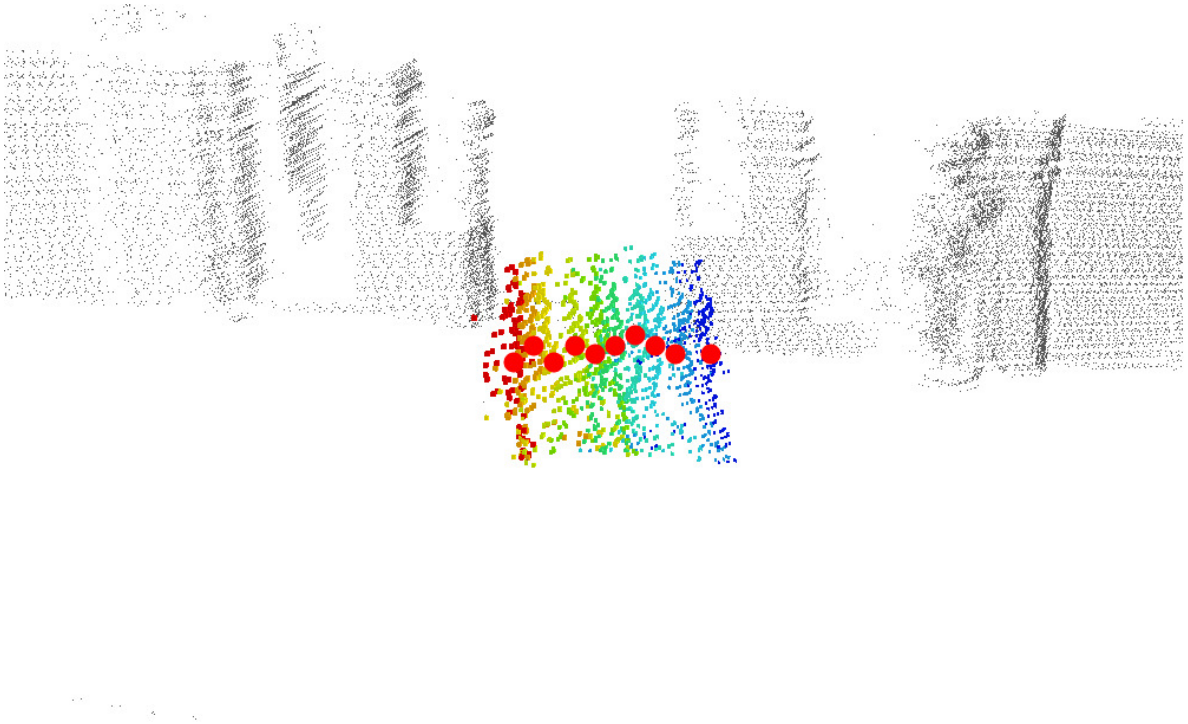


Figure 5.2: Moving pedestrian with visible centroids over the course of 10 frames - 1 second

5.2 Detecting dynamic objects

The main idea is that moving objects should have observable spatial differences between each centroid, example in Figure 5.2. In contrast, static objects should have only a tiny variance due to sensor flaws and movement of the ego vehicle carrying the LiDAR sensor. For each object represented by a cluster of points, we first calculate the centroid of the object at each time j as follows:

$$centroid_j = \frac{1}{|O \cap T_j|} \sum_{x \in O \cap T_j} x \tag{5.1}$$

where T_j is a PCL recorded at time j and so all points from $O \cap T_j$ are points recorded at time j that are part of a set O, representing the current cluster. $|O \cap T_j|$ denotes the cardinality of the set $O \cap T_j$. We then obtain the object's average speed:

$$average\_speed = ||\frac{1}{t-1} \sum_{i=1}^{t-1} centroid_{i+1} - centroid_i|| \tag{5.2}$$

where t denotes the number of time frames over which we calculate the centroids. We have decided to measure the norm of the average distance between consecutive pairs of centroids, as seen in Equation 5.2. All objects with an average movement speed higher than some threshold are considered dynamic.

According to research by Forde and Daniel [27], older pedestrians walk around 1 meter per second. To include rare cases such as a parent walking with a child or pushing a stroller, we have lowered this threshold to 0.6 meters per second, so if an object has an average speed

higher than 0.06 (meters per tenth of a second due to the LiDAR frequency), it is considered dynamic.

These steps do leave us with quite a lot of false positive cases; one such example can be seen in Figure 5.3. These cases occur because of the movement of the ego vehicle. LiDAR rays can reach new, previously occluded places, which now appear as dynamic. This problem is later addressed in Chapter 6.

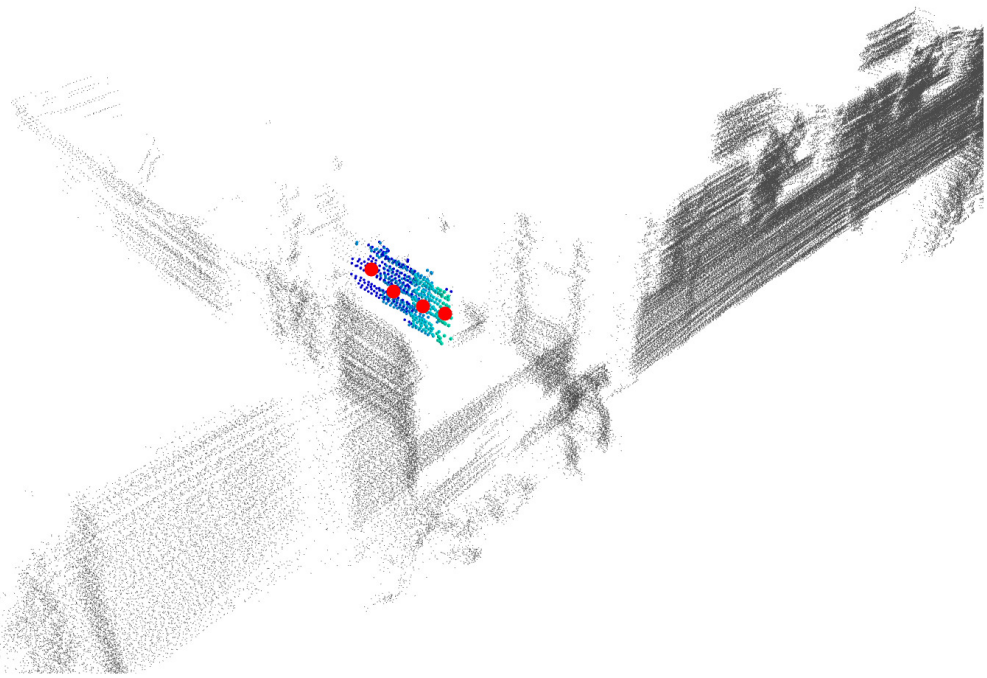


Figure 5.3: Falsely classified static object, that due to being occluded and because of the movement of the sensor appears as dynamic when considering centroids over time.

5.3 Post-processing

To eliminate false positives and increase precision, we have set additional criteria that each object must satisfy to be labeled as dynamic. The object must be seen the majority of times in the considered sequence; for example, it must have at least 6 individual centroids if we use a 10 PCL sequence. It naturally makes sense that objects that can not be observed well and are seen only a few times are more challenging to classify, and we are more prone to make a mistake on these objects.

Another criterion is the visibility of the ground. Dynamic objects usually move on an unobstructed flat surface and cannot go through other objects; therefore, the space the object passes should be empty once the dynamic object moves through. We use this assumption to check the ground under the observed object's first timestamp, on the condition that we see this object the majority of times; it is safe to assume that even a slow-walking pedestrian will move enough so that the ground behind him becomes observable.

Only objects that satisfy both criteria are labeled as dynamic. Consequently, we drop many false positive cases, although we incorrectly label some dynamic objects as static as a result. The output of this method can be seen in Figure 5.4.

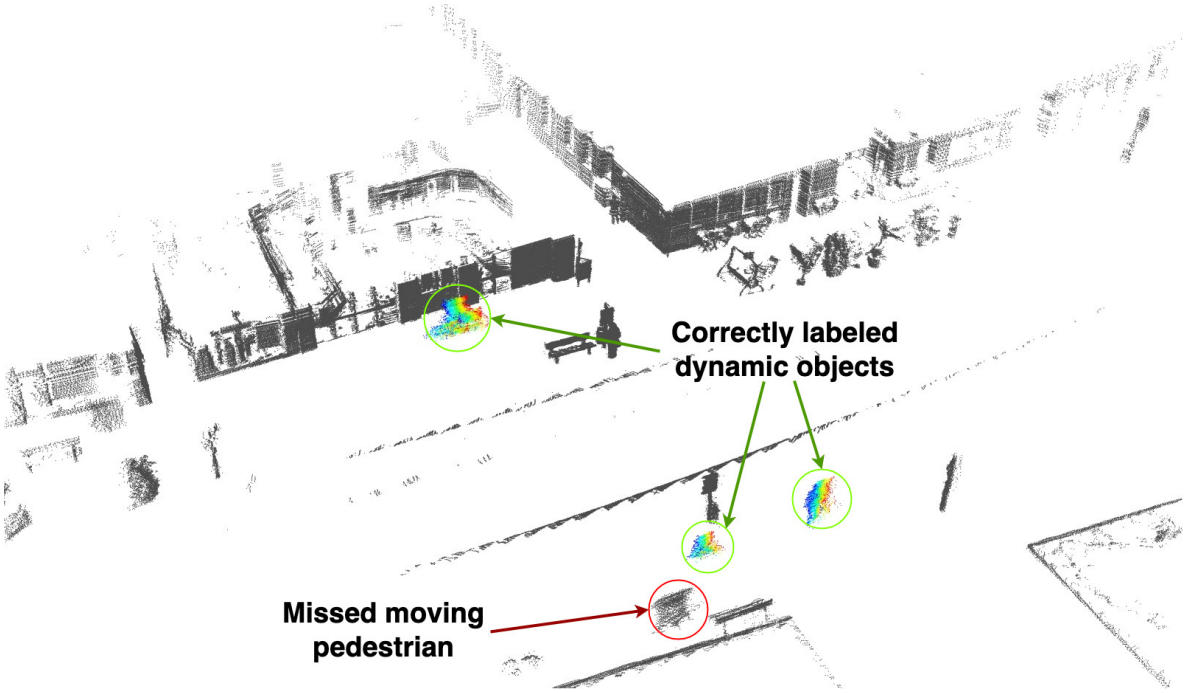


Figure 5.4: Result of the centroid-based method with post-processing on 10 synchronized PCLs. The cluster on top represents a pedestrian pushing a bike. The small cluster in the bottom middle is a child riding a small bike. This shows how flexible this method is.

5.4 Evaluation

This method shows significant improvement compared to the voxel-based method in Chapter 4. It is more customizable through its various hyperparameters, so this method can work on a whole different data set, which might use different LiDAR sensor.

We have also tested this method on the nuScenes[28] data set, which uses a 20 Hz 32-row LiDAR sensor instead of the 10 Hz 64-row LiDAR used for SemanticKITTI. An example can be seen in Figure 5.5.

The biggest downside of this method is that it cannot detect very fast-moving objects, such as cars. The reason is that because of their high speed, there is a large gap between their position at each frame, and so they are not clustered together; therefore, their centroids cannot be compared. This situation can be seen in Figure 5.6.

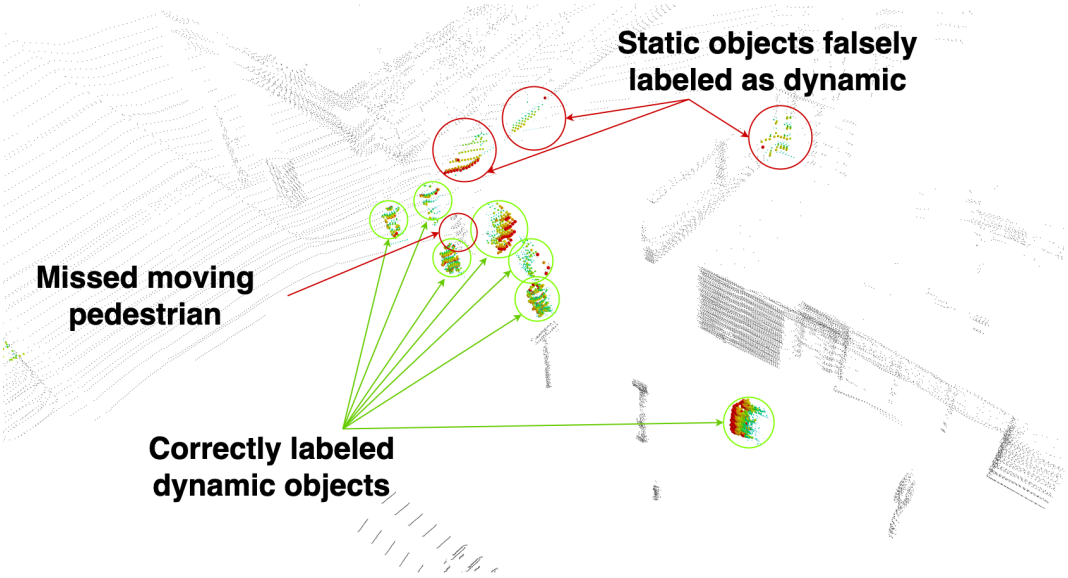


Figure 5.5: Result of the centroid-based method **without** post-processing on 10 synchronized PCLs on the nuScenes data set.

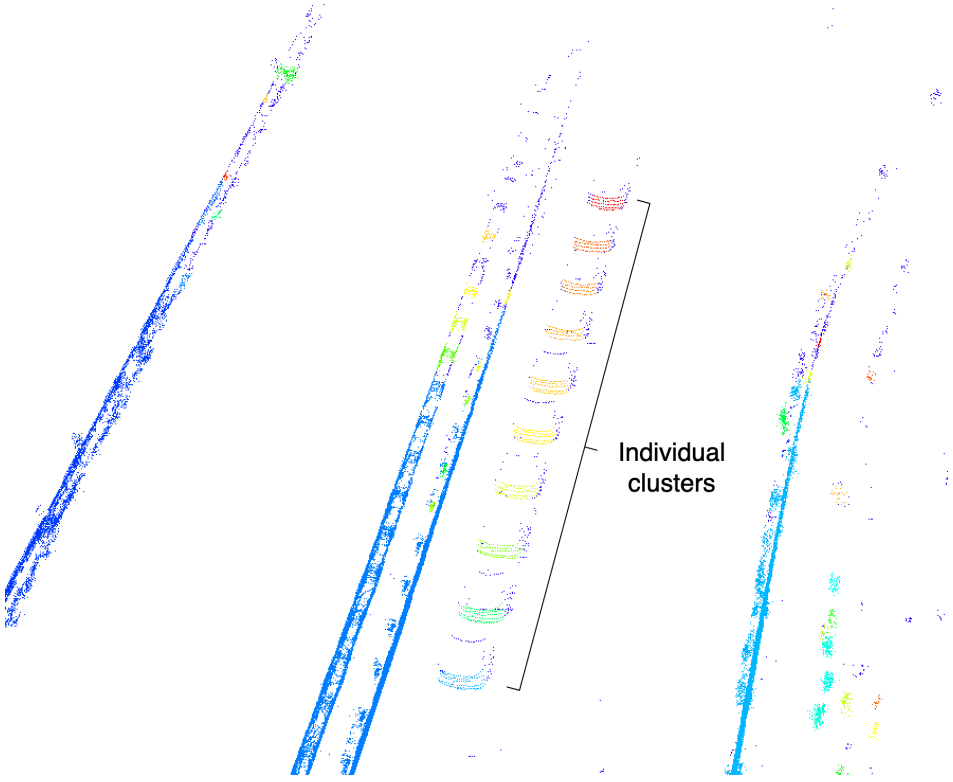


Figure 5.6: 10 synchronized PCLs where it can be seen, that the gaps between frames of a car are too big to be clustered together.

Chapter 6

Ray casting-based approach

The most prevalent error of previous methods is caused by the conjunction of the following factors: partially observable area and occlusions and the movement of the sensor. The ray casting method attempts to overcome this error with a relatively simple idea. We find all potentially moving clusters and disregard all we do not see well. Those that remain are considered dynamic, and those that are eliminated are considered undecided, still in line with the premise of high precision and a low number of false positives. In steps, this method does the following:

- Perform DBSCAN clustering on a single PCL
- Remove clusters of invalid sizes, described in Table 6.1
- For each remaining cluster, compute its centroid
- Look n frames into the future and/or the past if there are any points around the centroids area
- If not, the cluster is considered dynamic, and we perform ray casting to confirm; otherwise it is considered static

We perform two separate predictions, one for vehicles (cars and motorcycles) and one for pedestrians and cyclists, since they both have different densities and move at different velocities, and we merge both predictions at the end.

6.1 Pre-processing

First, we perform clustering on a single PCL that we wish to label. We then remove all objects that do not meet the size limits shown in Table 6.1. We have set the size limits based on our prior knowledge of the sizes of those objects. For the other parameters, we have used a grid search on a small representative subsample from multiple sequences to find the best combination of hyperparameters for both predictors. Grid search is a brute-force technique that evaluates all possible combinations of given hyperprarameters.

For cars, we ignore anything longer than 6 meters, wider than 5 meters, and taller than 2 meters. We also skip clusters shorter than 1 or smaller than 0.2 meters. For pedestrians, the maximum accepted length is 2 meters, as is the width, and the maximum height is 2.2 meters. These limits are set sufficiently high for rare cases where a pedestrian is pushing something, or two pedestrians are holding hands and are clustered together as one. The minimum height is 0.8 meters, and the minimum length is 0.3 meters.

For the clustering of vehicles, we use $\epsilon = 0.4$ and $min\_samples = 15$. For the clustering of pedestrians, we use $\epsilon = 0.3$ and $min\_samples = 40$. The reason behind these values is that cars are generally less dense than pedestrians. An example of car clustering can be seen in Figure 6.1.

Table 6.1: Hyperparameters for ray casting method. All size values are in meters. These parameters have been selected by a grid search optimization.

Parameter	Car	Pedestrian
ϵ	0.4	0.3
$Min\_samples$	15	40
Radius of ray	0.3	0.3
Maximum width	5	2
Maximum length	6	2
Minimum length	1	0.3
Maximum height	2	2.2
Minimum height	0.2	0.8
Number of frames to look into the future and/or the past	4	7

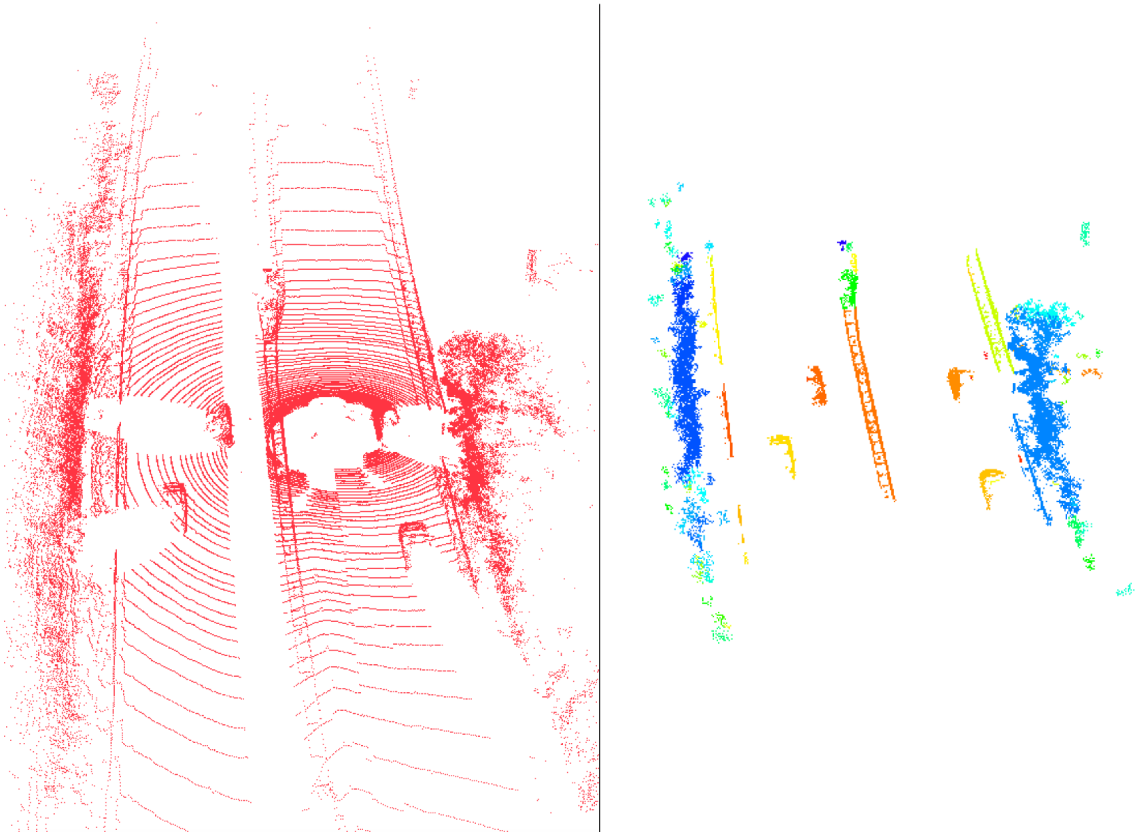


Figure 6.1: Left - single PCL; Right - the same PCL after clustering.

6.2 Detecting dynamic objects

We iterate through all clusters that have passed the size limits and compute their centroids. We look around the centroid's area in future and past PCL frames (if both are

available, if not, we use whichever one is), and if there are no points in at least one case, we suspect that this cluster might be dynamic, and we will perform ray casting at the end.

For the car predictor, we look at the PCL that is four frames into the future and the past, and the area around the inspected centroid is a sphere with a radius of 0.5 meters. For the pedestrian predictor, we look at the seventh frame into the future and the past and check 0.1 meters around the centroid. For all area checks, we use k-d trees, which is a data structure designed for fast searches based on coordinates, often used for the K-Nearest Neighbors (KNN) algorithm. This idea is visualized in Figure 6.2.

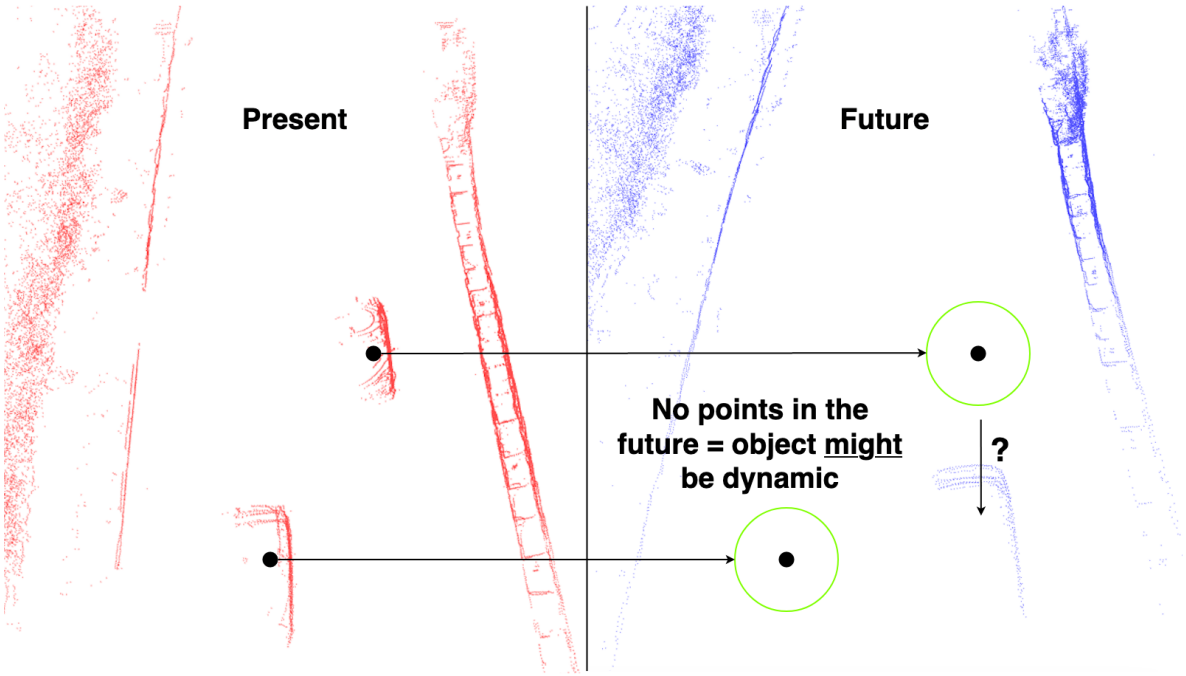


Figure 6.2: Left - present PCL, there are two cars whose centroids are depicted as black dots; Right - a PCL 4 frames in the future from the present PCL, the black dots are in the exact location as in the present PCL. The green circle around them represents the observed area where we check for any points. In this case, there are none; thus, both objects on the left are considered dynamic and will be further tested with ray casting. Note that we can see where the rightmost car has moved to in the future.

6.3 Post-processing

Lastly, for all the clusters that had no points around their centroids either in the future or in the past, we conduct ray casting to decide whether this has happened because the inspected cluster is occluded or because it truly is a dynamic object. We cast the ray in the future or past PCL, starting at the synchronized origin, where the LiDAR sensor would be in the real world, and look at whether the ray can reach the target destination (the inspected centroid) without hitting any points along the way. An example can be seen in Figure 6.3. If it cannot reach the observed target's centroid, we cannot decide whether the object is dynamic because it has been occluded and we set its class to undecided. If the ray does reach the target's centroid, then we have a situation where:

- In the present, we can see some object at a given position.

- In the future or past, there are no points in some small area around the object's centroid.
- The disputed area in the future or past is not occluded.

This means that the questioned object must have disappeared completely (unlikely), or it must have moved, meaning that it is dynamic. The width of this ray is one of the hyperparameters which contributes significantly to precision. The wider the ray, the less likely it will reach the target uninterrupted. For both predictors, we used a ray with a radius of 0.3 meters.

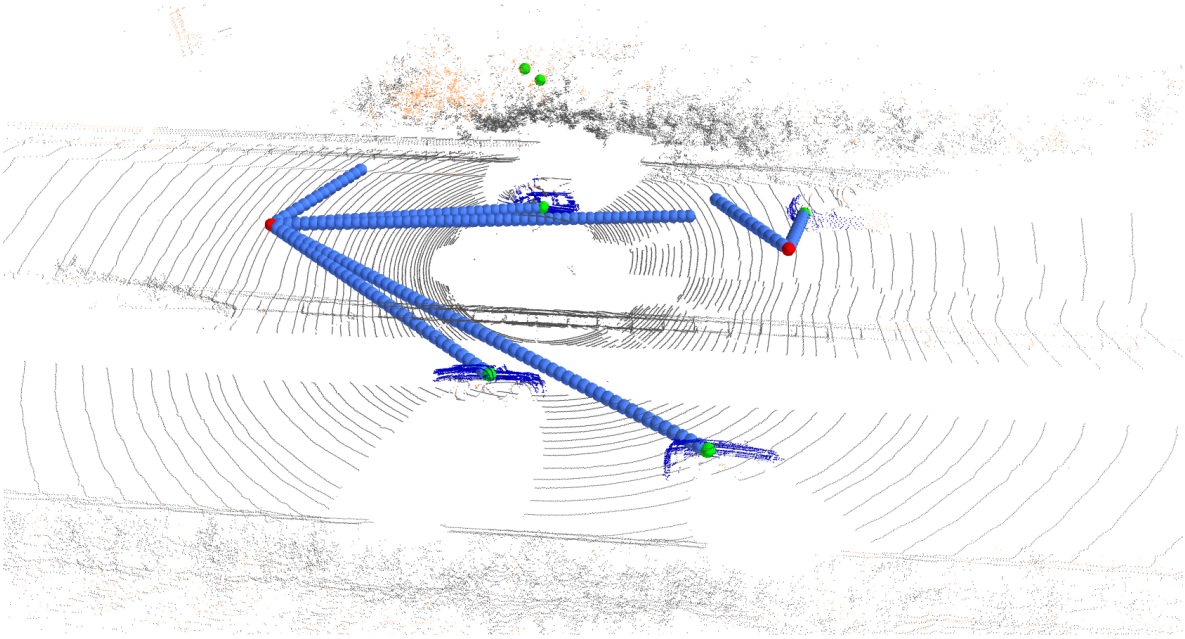


Figure 6.3: Example of ray casting from the future and the past. The red points represent the position of the LiDAR in the future and in the past. Green points represent centroids of objects which are considered dynamic. If a ray reaches the green point, the cluster containing this green point is labeled blue, meaning it is dynamic. Orange points represent the undecided class. Some rays stop before they have reached their designated green target. This is because in the future or the past, there are some points in the way of the ray.

6.4 Evaluation

This approach has proven to be the most resistant to false positives. With the numerous hyperparameters, it can be fine-tuned to find the best balance between precision and IoU. The average time it takes to label one scene is around two seconds, faster than previous methods, nevertheless still too slow for online use. We further evaluate this method in Chapter 7. An example of the result of this method can be seen in Figure 6.4.

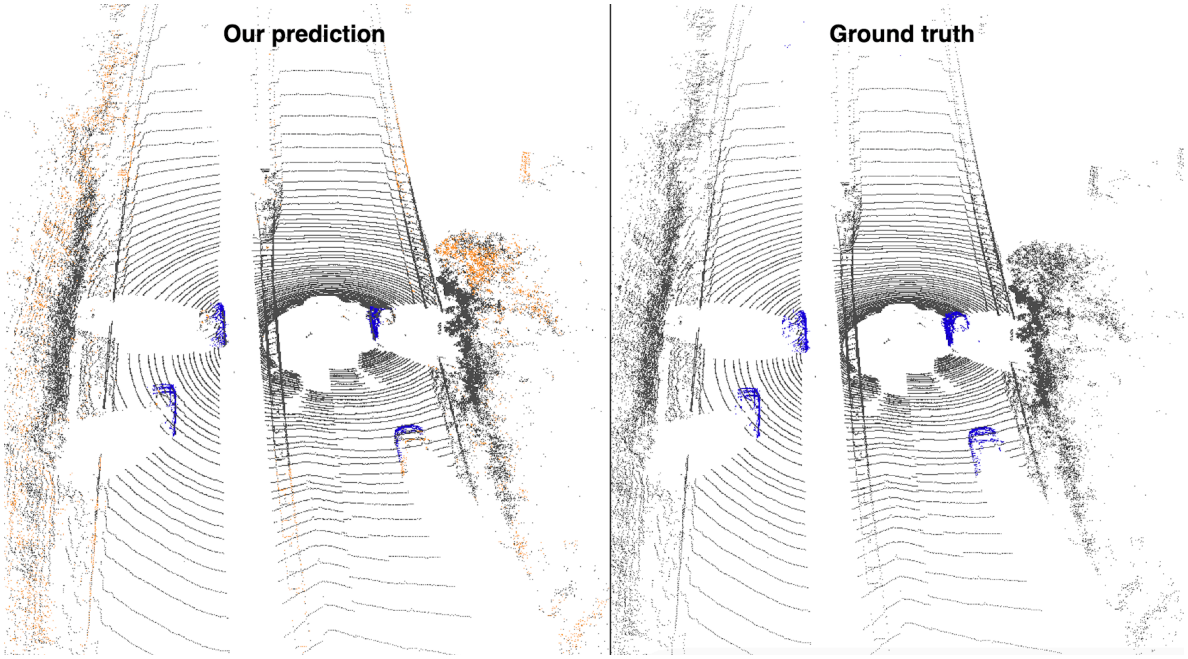


Figure 6.4: Left - prediction of our method, blue points are dynamic, dark points are static and orange points represent the undecided class; Right - the ground truth labels.

Chapter 7

Dynamic segmentation neural network

Semantic segmentation predicts a class label for each sensory point, be it a pixel in an image or a point in a PCL, allowing us to understand the scene.

Usually, there are many classes in a semantic segmentation method, such as pedestrian, car, road, sidewalk, and so on. However, we are interested only in dynamic and static segmentation. For this purpose, we have selected LMNet [3] for our experiments. This neural network is implemented in PyTorch [29], and uses residual images as its input, then performs a pixel-wise segmentation and propagates the labels back to the points in PCL. A schematics of the model architecture is depicted in Figure 7.1. The loss function it tries to minimize is the cross-entropy

$$L_{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{\mathrm{N}_c} y_i \log(\hat{y}_i) \tag{7.1}$$

where $\boldsymbol{y}$ is a vector of size N_c (number of classes, in our case we have 3: dynamic, static, undecided) with zeros at all positions but j, which contains one. This encodes categorical data where the position j decides which class this vector represents. The vector $\hat{\boldsymbol{y}}$ is the output of the forward pass. The last layer in LMNet is the softmax function, this means $\hat{\boldsymbol{y}}$ sums to one, and the number at each position represents the predicted "probability" of the class corresponding to that position.

As one can imagine, there are substantial differences in the number of dynamic points and the number of static points. This leads to an imbalance between classes and could cause the model never to predict the class with the lowest number of training examples. To compensate for this imbalance, we compute how many static and dynamic points there are using SemanticKITTI labels and weigh the loss based on the class ratio simply by multiplying the loss according to its class. There are over 296 times more static points than dynamic ones in our data set.

To obtain a residual image, we first synchronize each PCL of the observed sequence and then apply the spherical projection. This results in mapping all points from a 3D PCL space onto an image. We then assign to each pixel the range of its point computed by the L2 norm. We acquire residual images by taking the absolute values of pixel-wise subtraction of two range images. The network can take an arbitrary number of residual images; according to the authors, the best results are achieved by using eight residual images.

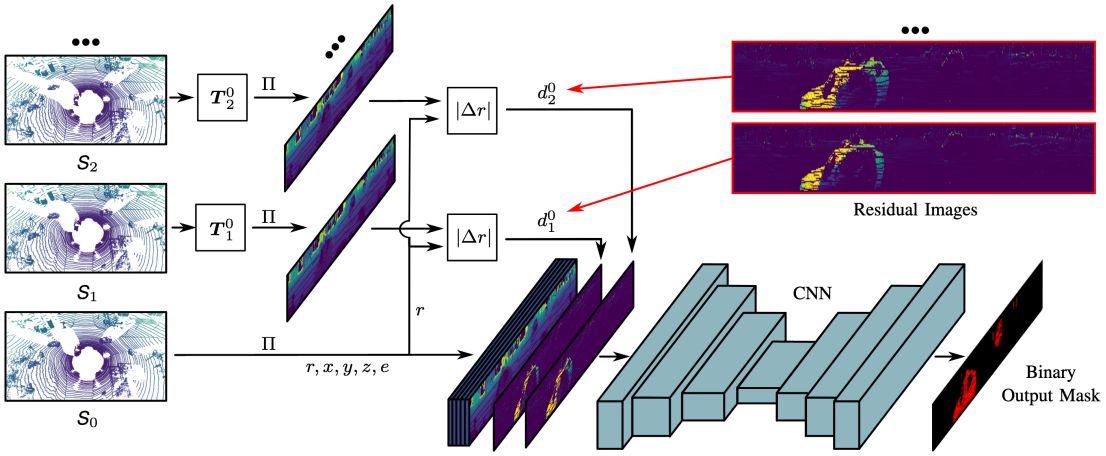


Figure 7.1: A schematics of a LMNet architecture, taken from [3].

To evaluate the usability of our method, we used it to generate labels for the KITTI data set [7]. We then use those labels to train LMNet, which also works as a baseline in a Motion Object Segmentation (MOS) competition. LMNet was initially trained for over 150 epochs on SemanticKITTI labels using sequences 0 to 7, 9, and 10. Sequence 8 was used for the validation set, and the remaining sequences 11-21 were used as the testing set.

However, the labels for sequences 11-21 are not made public and are used for evaluation in the MOS competition. Due to this, we have to use one of the labeled sequences as a test set. We used sequence 10 as the validation set, sequence 8 for testing, and all other sequences are used for training, including sequences 11 to 21, which we label using our unsupervised method.

We compare multiple training runs, which vary in the percentage of SemanticKITTI Ground Truth (GT) labels. In particular, we tried runs with 5%, 10%, 20%, 30%, 40%, and 100% of GT labels, where 5% means that we have used 5% of the available GT labels for the training set, and the rest are labels generated by our method. 0% is an unsupervised run, training only on our automatically-generated data. The system is also explained in Figure 7.2.

We have also trained one model only on the provided SemanticKITTI labels (we call it original) as a baseline; therefore, this model only uses sequences 0 to 7, 9, and 10 as its training data.

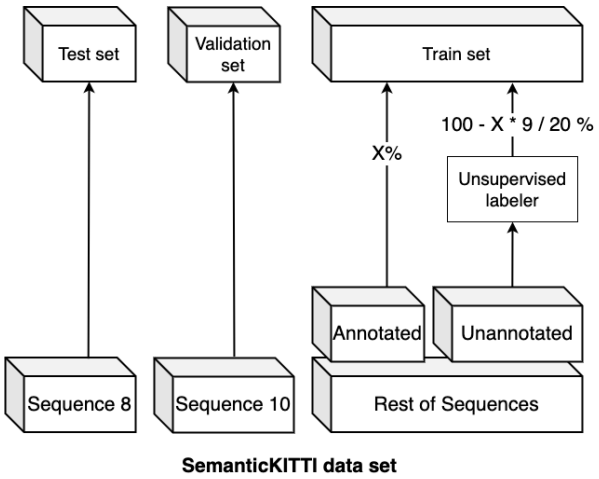


Figure 7.2: Schematic explaining the composition of the training set.

We do this to simulate a real-world scenario where a team developing a solution needs to know whether more hand-labeled data will noticeably improve their performance.

7.1 Implementation details

All the presented runs shared the same hyperparameters, which can be seen in Table 7.1. We have trained the models on the Research Center for Informatics cluster, each running on one NVIDIA A100 40GB GPU, and it took all runs but the original over ten days to finish 180 epochs and eight days for the original. The original run converged in epoch 127, while the others converged before the 90th.

Table 7.1: Hyperparameters used for training the LMNet.

Parameter	Value
Loss function	Cross-entropy
Maximum epochs	180
Warm-up epochs	1
Momentum	0.9
Optimizer	SGD
Learning rate decay	0.99
Weight decay	0.0001
Batch size	2
Epsilon weight	0.001
Residual	True
Num. of residual images	8

7.2 Evaluation

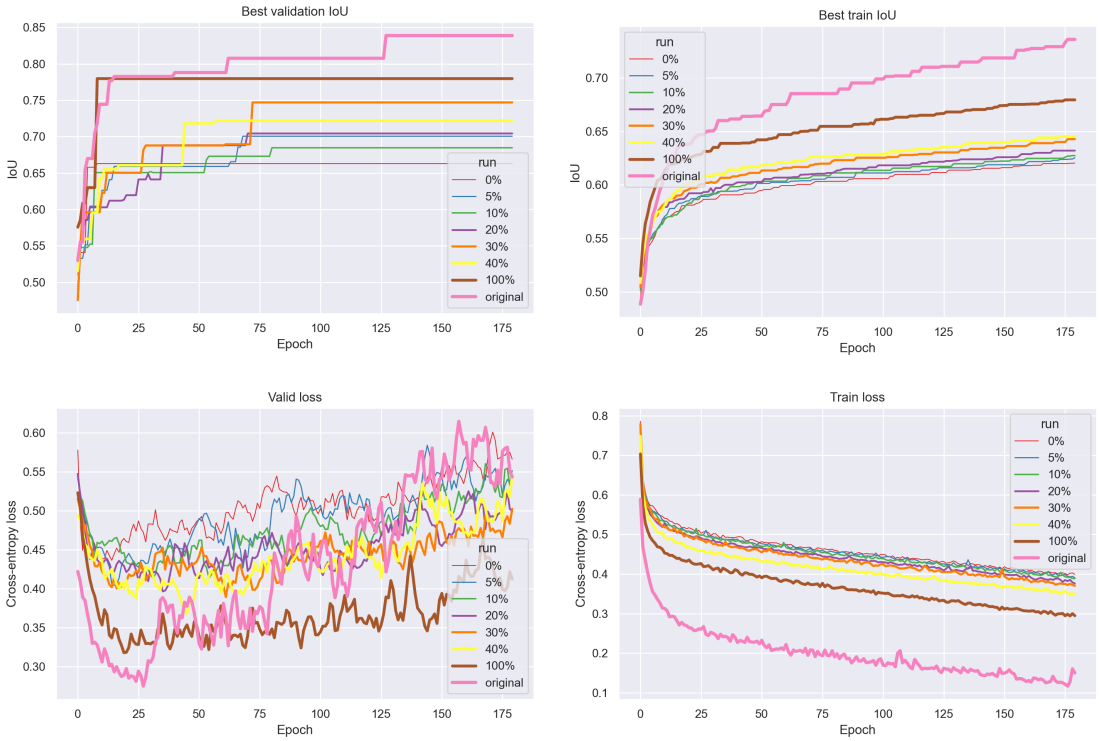


Figure 7.3: Validation and training losses and IoUs of all runs over 180 epochs.

We let all our runs train on 180 epochs, and we store the model with the best IoU on the validation set, the development of training can be seen in Figure 7.3. The validation data are kept separate from the training data, this is to prevent over-fitting. Over-fitting results in model performing well on the training data - the data it has seen, but fails to generalize and performs poorly on new data (e.g., test set).

Because we choose the model's weights based on its performance on the validation set, we still need an extra test set to get a good estimate of our model's performance. The results of the testing set evaluation can be seen in Figure 7.4 and Table 7.2.

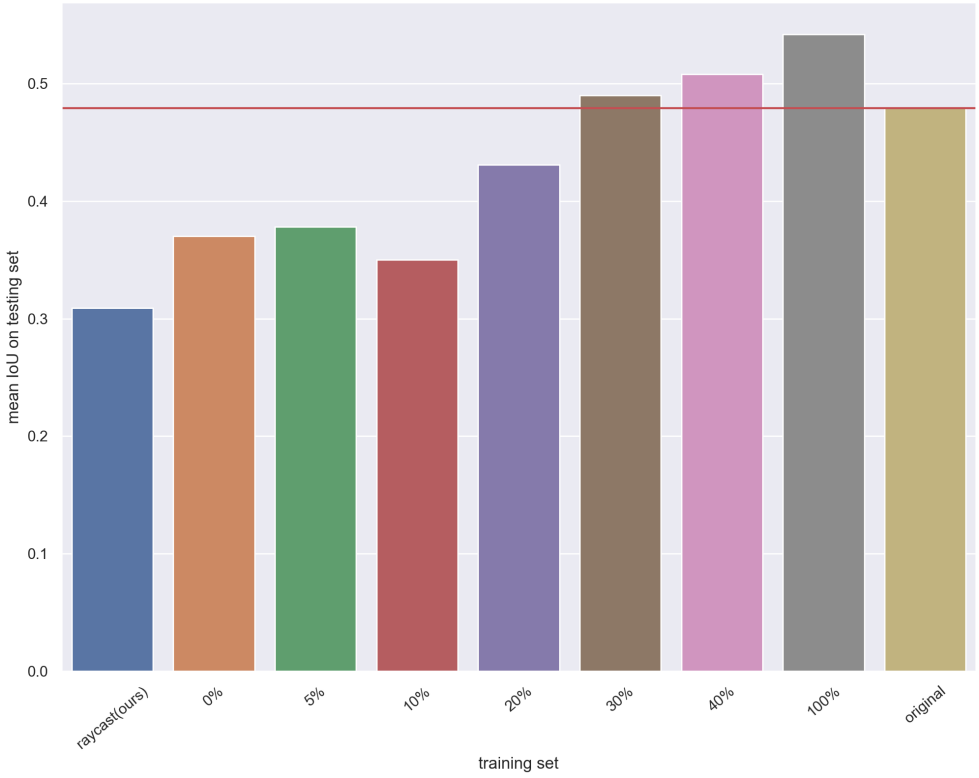


Figure 7.4: Mean IoU on the moving class, averaged over the whole test set. Only the moving class is measured here, as everything that is not moving is static, and so better IoU on the moving class equals better IoU on the static class. All evaluated runs are neural networks, except for the raycast run, which is the evaluation of our ray cast method.

Table 7.2: Evaluation of the model's performance on the test set.

Training set	Mean IoU on the moving class
raycast (ours)	0.309
0% (unsupervised)	0.37
5%	0.378
10%	0.35
20%	0.431
30%	0.490
40%	0.508
100%	**0.542**
original [3]	0.479

As one can see, we have surpassed the original model's performance in runs 30%, 40%, and 100%. We can see that the best results were achieved using all of the given GT labels and the rest of ours, which we would expect. However, the biggest value here is that one can reach similar results using only one-third of hand-labeled data and let our algorithm label the rest.

It is important to note that all runs except the original have twice as many training data. This results in worse training loss and train IoU, as those models have those metrics computed over more data. However, those models generalize better as they have seen more examples.

We planned to participate in the MOS competition; however, because of the introduction of the class undecided, we did not meet the technical requirements. Due to this, we had to evaluate our models ourselves and reserve part of the data for the test set, which we would normally use as a validation set.

Chapter 8

Correspondences

A standard reoccurring error while inferring a segmentation neural network is a small group of points switching their predicted classes from frame to frame. We want to improve the model's performance with the assumption of spatial similarity - two same points at two consecutive time frames (with a fast enough frame rate) should be classified similarly. This requires using an already pre-trained segmentation model and some model for matching corresponding points between two frames. The schematic can be seen in Figure 8.

We tried three approaches for matching points: voxelization, 1-nearest neighbor, and scene flow. This new self-supervised loss accompanies the original cross-entropy loss. At each step of an epoch, we perform two forward passes, obtaining two outputs. One from the training set for which we have GT labels and one from the test set for which we have no GT labels. First, we calculate the unsupervised loss on one batch and perform optimization via backpropagation. Then we calculate the original loss on a different batch and again go through backpropagation.

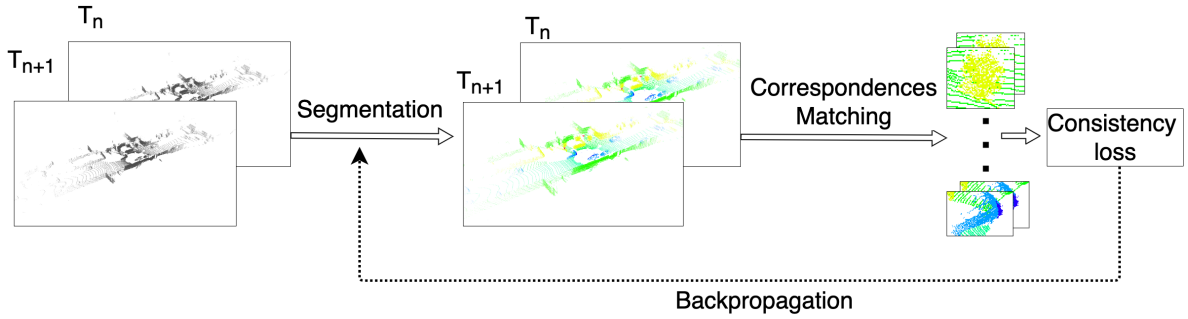


Figure 8.1: Diagram for self-supervised loss.

8.1 Voxelization

Returning to the idea from 4.1 that we can split our observed space into voxels, synchronize two consecutive PCLs, and go through each voxel, watching how the predictions differ between the two frames. The criterion we have chosen is to find all voxels whose points have uniform predictions, though they need not be the same across times.

Suppose the predictions are the same at time t and time $t+1$. In that case, we reinforce those probabilities at both times with a cross-entropy loss, indicated in Equation 7.1, where y_i is a binary indicator if a class i is the predicted class. Minimizing this loss will push the probabilities closer to 1.

If the predictions differ in time, we decrease the probabilities at both times through a negative cross-entropy loss, also called the log-likelihood

$$L_{LL}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{\mathrm{N_c}} y_i \log(\hat{y}_i) \tag{8.1}$$

minimizing this loss will push the probabilities closer to 0.

8.2 1-nearest neighbor

Another way to match close points between two frames is to use the nearest neighbor algorithm. This assigns each point from time $t + 1$ its closest point from time t. We do not consider points that are further than 1 meter from their closest neighbors. And then we carry on with the losses in the same way as before: for all pairs that are equal in the predicted classes, we apply the cross-entropy loss, shown in Equation 7.1, and for all pairs that are not equal, we use the log-likelihood loss, defined in Equation 8.1.

8.3 Scene flow

Scene flow is a vector representation of a motion between two consecutive PCLs. It assigns each point from one frame its corresponding point in the next frame. Mittal et al. [18] propose a self-supervised neural network for estimating scene flow based on FlowNet3D [4] architecture, which would provide us with an estimate of a one-to-one translation.

We have tried to train this model on our dataset; however, we have been unable to reach similar results as the authors, partially because self-supervised methods require much more data to train, as their loss functions are not as tight as in supervised methods, where we explicitly maximize the probability of the correct result. Therefore, we could not use this approach for our task, and we only evaluated voxelization, and 1-nearest neighbor approaches.

8.4 Evaluation

First, we evaluate how well our correspondence matching performs on multi-class semantic data, where we have more than two classes. The results can be seen in table 8.1.

Table 8.1: Evaluation of correspondences correction for 1-nearest neighbor on a few classes. Values measured in %.

Class	Vegetation	Vehicles	Pedestrian	Cyclist	Traffic Sign
Correct matches	91.9	89.3	88.1	93.3	82.1

As we can see, in most cases, it performs very well, and thus, in theory, it should help increase our IoU. In reality, it did not improve our model. In figure 8.2 we can see the model's performance on the validation set and the train set throughout a few epochs. The model we have used is an already pre-trained LMNet neural network.

We can see a considerable fall in performance on both sets; this is likely because the model does not perform sufficiently well, and the wrong classifications are propagated through the correspondences. This approach ultimately proved to be more complicated than anticipated and would require more time to incorporate.

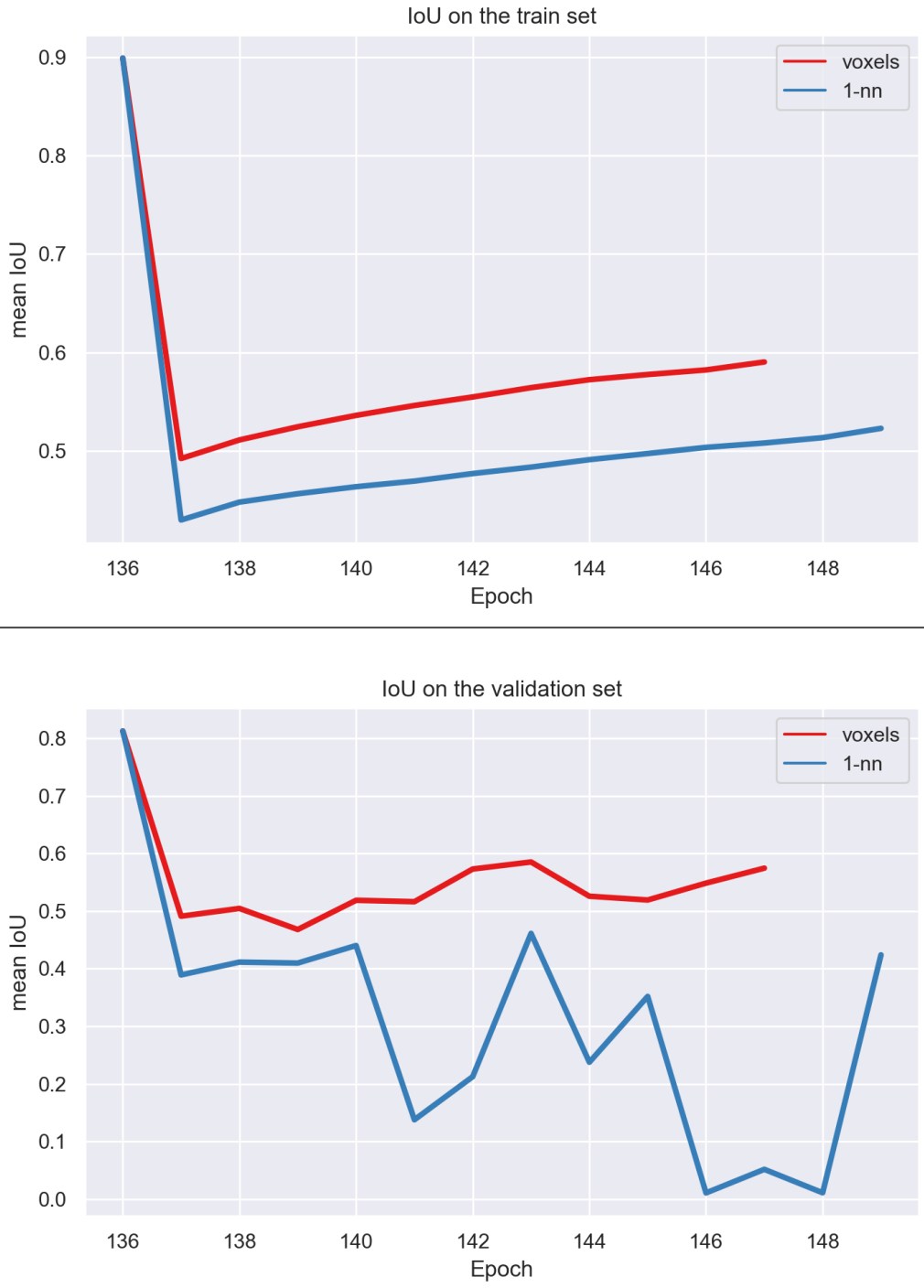


Figure 8.2: Top - mean IoU on the train set; bottom - mean IoU on the validation set.

Chapter 9

Conclusion

We have explored and evaluated algorithms for removing ground points from the PCL. We have examined three approaches for detecting dynamic objects using unsupervised methods and used the best one to generate labels that we then used to train a CNN. Our method combines our prior knowledge of dynamic objects (e.g., size, minimum speed) and exploitation of the dynamics of objects. We then evaluated multiple experiments showing the impact of our data on the supervised model's performance. We have shown that given enough unlabeled data, one can reach better results by only having to annotate one-third of the original data set and using our method to label the rest.

We experimented with a self-supervised loss that could help further improve the model's performance, but the method did not achieve any improvement.

Further improvement could be made by incorporating a different method to obtain segmentation. The knowledge of the ground's position could be used to filter out falsely labeled points.

Bibliography

[1] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "Polarnet: An improved grid representation for online lidar point clouds semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[2] Y. Wang, T. Shi, P. Yun, L. Tai, and M. Liu, "Pointseg: Real-time semantic segmentation based on 3d lidar point cloud," 2018.

[3] X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss, "Moving Object Segmentation in 3D LiDAR Data: A Learning-based Approach Exploiting Sequential Data," *IEEE Robotics and Automation Letters(RA-L)*, 2021.

[4] X. Liu, C. R. Qi, and L. J. Guibas, "Flownet3d: Learning scene flow in 3d point clouds," *CVPR*, 2019.

[5] M. Sualeh and G.-W. Kim, "Dynamic multi-lidar based multiple object detection and tracking," *Sensors*, vol. 19, no. 6, 2019.

[6] 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.

[7] 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)*, pp. 3354–3361, 2012.

[8] B. Ravi Kiran, L. Roldao, B. Irastorza, R. Verastegui, S. Suss, S. Yogamani, V. Talpaert, A. Lepoutre, and G. Trehard, "Real-time dynamic object detection for autonomous driving using prior 3d-maps," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[9] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 163–169, 2016.

[10] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde, "Fast LIDAR-based road detection using fully convolutional neural networks," in *2017 ieee intelligent vehicles symposium (iv)*, pp. 1019–1024, IEEE, 2017.

[11] P. Chu, S. Cho, S. Fong, and K. Cho, "Enhanced ground segmentation method for Lidar point clouds in human-centric autonomous robot systems," *Human-centric Computing and Information Sciences*, vol. 9, 12 2019.

[12] I. Bogoslavskyi and C. Stachniss, "Efficient Online Segmentation for Sparse 3D Laser Scans," *Photogrammetrie - Fernerkundung - Geoinformation*, vol. 85, pp. 41–52, 12 2016.

[13] Z. Shen, H. Liang, L. Lin, Z. Wang, W. Huang, and J. Yu, "Fast Ground Segmentation for 3D LiDAR Point Cloud Based on Jump-Convolution-Process," *Remote Sensing*, vol. 13, no. 16, 2021.

[14] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," *CoRR*, vol. abs/1812.04244, 2018.

[15] J. Yin, J. Shen, C. Guan, D. Zhou, and R. Yang, "Lidar-based online 3d video object detection with graph-based message passing and spatiotemporal transformer attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[16] H. Tian, Y. Chen, J. Dai, Z. Zhang, and X. Zhu, "Unsupervised object detection with lidar clues," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5962–5972, June 2021.

[17] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," 2017.

[18] H. Mittal, B. Okorn, and D. Held, "Just go with the flow: Self-supervised scene flow estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[19] D.-H. Lee, "Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks," *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[20] X. Zhu, H. Zhou, T. Wang, F. Hong, Y. Ma, W. Li, H. Li, and D. Lin, "Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9934–9943, 2021.

[21] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han, "Searching efficient 3d architectures with sparse point-voxel convolution," in *European conference on computer vision*, pp. 685–702, Springer, 2020.

[22] K. Lunda, "Light source measurement."

[23] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[25] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[26] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996.

[27] A. Forde and J. Daniel, "Pedestrian walking speed at un-signalized midblock crosswalk and its impact on urban street segment performance," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 8, no. 1, pp. 57–69, 2021.

[28] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.