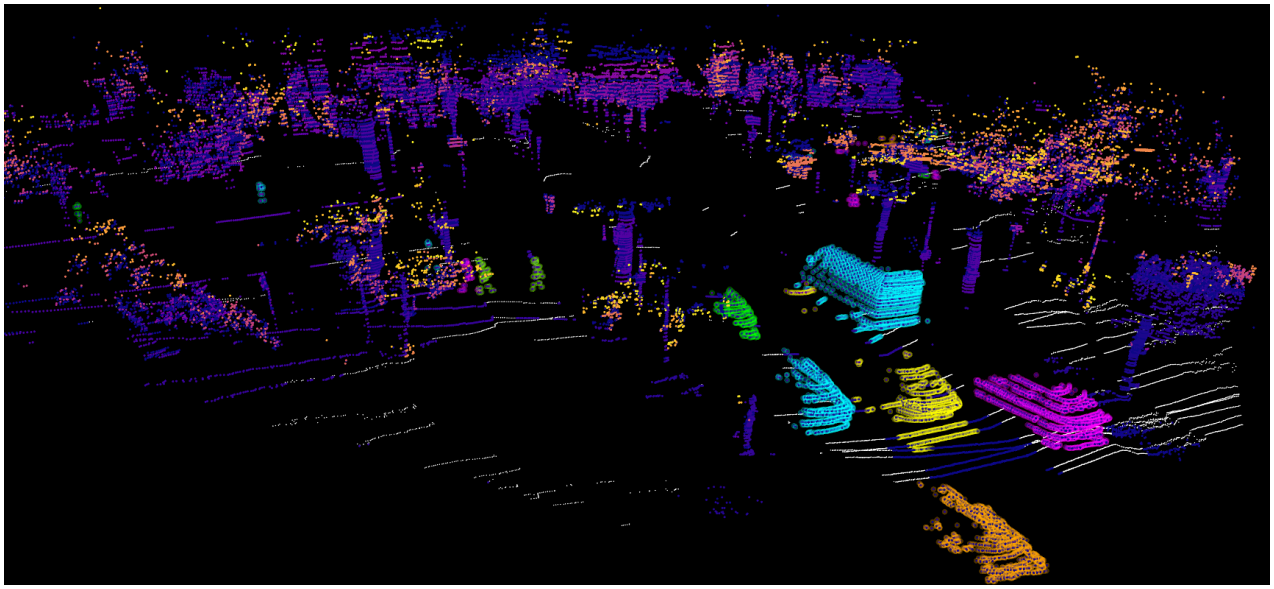
CHALMERS
UNIVERSITY OF TECHNOLOGY



Robust Model-Based Clustering Techniques for Non-Uniform LiDAR Point Clouds via Range Image Transformations

Adaptive Clustering Approaches for Multi-Layer Non-Uniform LiDAR Data from Luminar Iris Sensors

Master's thesis in Electrical engineering

OSAMA AL SHEIKH ALI & SIZHE CHEN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

Robust Model-Based Clustering Techniques for Non-Uniform LiDAR Point Clouds via Range Image Transformations

Adaptive Clustering Approaches for Multi-Layer Non-Uniform LiDAR Data from Luminar Iris Sensors

OSAMA AL SHEIKH ALI & SIZHE CHEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Robust Model-Based Clustering Techniques for Non-Uniform LiDAR Point Clouds via Range Image Transformations
Adaptive Clustering Approaches for Multi-Layer Non-Uniform LiDAR Data from Luminar Iris Sensors
OSAMA AL SHEIKH ALI & SIZHE CHEN

Zenseact Supervisors: Staffan Nilsson, Domenic Geiseler & Saman Mahmoodi
Chalmers Supervisor: Lars Hammarstrand, Department of Electrical Engineering
Examiner: Mats Viberg, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Clustering results from the developed algorithms where cars are have different colors indicating different classes.

Typeset in LaTeX
Gothenburg, Sweden 2025

Robust Model-Based Clustering Techniques for Non-Uniform LiDAR Point Clouds via Range Image Transformations
Adaptive Clustering Approaches for Multi-Layer Non-Uniform LiDAR Data from Luminar Iris Sensors
OSAMA AL SHEIKH ALI & SIZHE CHEN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous driving is a rapidly advancing field that is reshaping the future of transportation by promising improved traffic flow, reduced energy consumption, safer roads, and the elimination of human intervention, thereby saving time. An essential part of autonomous driving is a strong perception system such cameras and radar, which are key for safety features like lane keeping, adaptive cruise control, and emergency braking. However, LiDAR technology stands out being a central part of the perception system due to its ability to provide high-quality 3D spatial information even during poor lighting conditions. A specific type of modern LiDAR sensor (e.g., Luminar Iris) introduces non-uniformity in point cloud distribution by offering adaptive resolution. This feature provides more details for the area of interest, while reducing unnecessary data in less relevant areas such as the sky.

However, this non-uniformity of the LiDAR data introduces challenges for classical clustering algorithms, which work under the assumption of a uniformly distributed LiDAR point cloud (i.e., fixed sensor angular resolution). Consequently, these algorithms will struggle with over and under-segmentation problems when the distance between objects varies. In this thesis, two robust model-based clustering algorithms are developed and designed to cluster non-uniform LiDAR point clouds. The methods build on classical breakpoint detection and range image-based clustering approaches to handle varying point densities while still being computationally efficient. Furthermore, two ground removal approaches are developed and evaluated along with a suggestion of two evaluation methods, visual and quantitative evaluation on a dataset of Luminar point clouds. The results demonstrate better clustering performance of the two developed algorithms compared to a baseline algorithm, thus resulting in fewer over-segmentation and under-segmentation errors. Moreover, a thorough analysis of computational is provided. In summary, this research resulted in a stable and reliable clustering performance across diverse scenarios, which helps to realize safer and more predictable autonomous driving without relying on large-scale labeled data for training, as in deep learning models.

Keywords: Autonomous Driving, Clustering, LiDAR, Non-uniform Point Cloud, Model-based algorithms, Breakpoint Detection, Range Image, Over-segmentation, Under-segmentation, Luminar Iris

Acknowledgements

We would like to sincerely thank our Chalmers examiner, Prof. Mats Viberg, and supervisor Prof. Lars Hammarstrand, for the constant support, patience, and valuable feedback throughout this thesis. Your guidance kept us on track and helped us grow both technically and personally.

We are super grateful to Zenseact for giving us the chance to work on this exciting and real-world relevant project. Special thanks to our Zenseact's mentors, Staffan Nilsson, Domenic Geiseler & Saman Mahmoodi, whose insights helped shape our understanding and challenged us to think beyond the obvious. We appreciate the support and shared expertise from all team members. Special thanks to our engineering manager, Niloufar Boustani, whose elegance and warmth lifted our spirits!

To our families and friends, thank you for always being there. Whether it was a few kind words, late-night calls, or just your belief in us, it meant everything. Life is just better with you around.

We are grateful to each other as thesis partners. Six months is no short journey, and working together has been an adventure filled with brainstorming sessions, collaborative problem-solving and shared all-nighters with encouragement. Despite different working styles, we complemented each other excellent to complete well.

One personal note from Sizhe: Thanks for the good weather and the sustained energy that my thesis partner and I maintained. Though spring tried to slow us down with sniffles, we powered through and enjoyed the process.

Lastly, we're grateful to Chalmers University of Technology for creating a space where ideas and curiosity are encouraged to grow. We hope that readers will find this project as interesting and meaningful as we do that emerged from our collaboration.

Osama Al Sheikh Ali & Sizhe Chen, Gothenburg, 2025-06-24

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis, listed in alphabetical order:

ABD	Adaptive Breakpoint Detection
AC	Angle-based Cluster
ADAS	Advanced Driver Assistance Systems
APD	Avalanche Photodiode
BFS	Breadth-First Search
CalibRef	Calibration Reference
DBD	Dual Breakpoint Detection
DBSCAN	Density-Based Spatial Clustering of Application with Noise
DC	Distance-based Clusters
DL	Deep Learning
HBAC	Hybrid Breakpoint and Angular Clustering
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Application with Noise
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
ML	Machine Learning
MC	Motorcycle
RGAC	Range-Guided Adaptive Clustering
ToF	Time of Flight
XCVR	Transceiver

Contents

List of Figures

List of Tables

1

Introduction

A major global problem is traffic accidents which are the direct cause of over 1.2 million deaths per year. These accidents do not only cause a devastating human toll but also impose a tremendous economic burden with an estimated cost surpassing 1.8 trillion USD between 2015-2030 [14]. Autonomous driving technology offers the potential to improve transportation in a multitude of ways, including reduction of energy consumption, prevention of future accidents, and improvement of traffic flows. Furthermore, autonomous vehicles will free up time and mental capacity for the driver [34], [47].

A necessity to ensure the safety of autonomous vehicles is a precise and trustworthy perception system that senses the environment around the vehicle. An example of such system is Light Detection and Ranging (LiDAR), which offers high-precision 3D spatial information [43]. LiDAR technology is robust in challenging circumstances, such as low-light conditions, by continuously and accurately measuring distances the distances to surrounding objects around us unlike camera systems which often struggle in such circumstances.

1.1 Background

There are different levels of LiDAR technology including single-layer, multi-layer, and 3D laser scanners. Single-layer laser scanners use just one beam, while multi-layer scanners provide basic 3D view by using two and seven layers. On the contrary, 3D laser scanners send out eight or more layers which provide a much richer 3D representation. The LiDAR sensor used in this thesis belongs to this latter category [29].

The LiDAR sensor used in this study is the Luminar Iris which produces non-uniformly distributed point cloud as opposed to tradition LiDAR sensors [35]. The benefit of this non-uniformity is the adaptive resolution, allocating more detail to regions of in interests (e.g., objects ahead at specific height) while minimizing unnecessary data in less crucial areas [35]. However, with this non-uniform feature comes unique challenges for object detection algorithms. Using traditional clustering methods will therefore lead to over-segmentation (splitting a single object into multiple clusters) or under-segmentation (combining multiple objects into a single cluster) due to the assumption that the point cloud has a uniform distribution.

In addition, fixed-parameter algorithms struggle to maintain consistent performance across different distances when point cloud density varies because parameter values that work for a close-range object may not work at all for a long-range object. In an autonomous driving environment, this inconsistency can lead to dangerous detection failures and accidents.

Another challenge that needs to be solved is to effectively process large amounts of point cloud data acquired from the LiDAR sensors, and then accurately cluster various types of objects at a rapid pace in diverse complex scenarios. In recent years, machine learning (ML) and specifically deep neural networks, have become a mainstream approach in the field of object detection and have achieved remarkable results on standard (i.e., uniformly distributed point clouds) datasets. However, these deep learning-based methods have inherent limitations in terms of training data and inconsistencies.

1.2 Main Research Questions

The focus of this thesis is to implement and analyse the performance of model-based clustering techniques on real world non uniform LiDAR point clouds. Example of challenges to consider are occlusions, varying point densities and cluttered environments. The research questions are:

1. How can classical (non deep-learning) algorithms effectively cluster objects in non-uniform LiDAR point clouds across various driving environments such as urban, highway, rainy and clear weather scenarios?
2. What are the trade-offs between clustering accuracy and computational efficiency when using model-based clustering techniques?

1.3 Purpose

The automotive industry is striving towards autonomous driving technology that is safe. Therefore effective and accurate target detection systems are a core requirement. This is particularly emphasized as the mission of Zenseact, which is committed to developing cutting-edge autonomous driving solutions.

At the center of this transformation is LiDAR technology. For Zenseact, LiDAR is not just an augmentation technology, but a key element in enabling safe and predictable autonomous driving, as it enables accurate depth estimation, object classification, and detection.

While many recent algorithms rely on deep learning (DL), these methods frequently necessitate large-scale labeled datasets and may struggle to generalize to new settings or unknown objects. The purpose of this thesis is to investigate an alternative approach that does not rely on a labeled dataset, deep learning, and to improve the generalization of LiDAR-based perception systems, particularly when dealing with non-uniform LiDAR sensors.

1.4 Limitation

This thesis emphasizes traditional clustering techniques and does not explore data-driven alternatives such as neural networks or sensor fusion with cameras. No exploration of the classification of recognized objects (e.g., identifying cars, pedestrians, or traffic signs) is done in this thesis, as no semantic information, such as object shape, size, or category, is available in the data.

The clustering algorithm is written in Python, which is not conducive to real-time applications when compared to using C++, for instance. Due to data collection methods being resource-intensive, the algorithm developed is evaluated on a limited dataset. Additionally, the algorithm is primarily optimized for the Luminar Iris LiDAR sensor and may need adaptations to work with other sensors.

1.5 Main Contributions

This research makes multiple contributions to the field of autonomous vehicle perception in several key ways. First, an efficient ground removal process is proposed to remove ground points that can wrongly connect objects that are spatially separated objects. Second, two robust non-DL clustering algorithms capable of reliably segmenting objects in a variety of environmental circumstances and distances, are created to assure the safety and dependability of self-driving systems. The two methods are called Hybrid Breakpoint and Angular Clustering (HBAC) and Range-Guided Adaptive Clustering (RGAC), respectively. Additionally, this thesis provides a systematic method for evaluating clustering performance on real-world data, including visual inspection and specialized over-segmentation and under-segmentation metrics.

Additionally, this thesis addresses the unique challenges of processing point cloud data from multi-layer LiDAR sensors by extending traditional breakpoint detection and clustering methods to 3D LiDAR data. This approach combines and improves breakpoint detection and range image-based clustering techniques to accommodate the characteristics of non-uniform point clouds while maintaining the computational efficiency and providing strong indications of possible real-time performance.

Finally, this research provides insights into the trade-offs between detection accuracy and computational efficiency in LiDAR point cloud clustering and proposes new methods to optimize these trade-offs.

Although there has been a great deal of research in LiDAR point cloud processing, the research on clustering algorithms specifically targeting the characteristics of modern non-uniform LiDAR sensors, such as Luminar Iris, is still limited. This innovation not only provides a reliable method of object detection for Luminar sensor applications but also opens up new possibilities for clustering algorithms to be used in modern LiDAR systems without utilizing deep learning.

1.6 Thesis outline

Following this introduction, the remaining part of this thesis is organized as follows. Sect. 2 provides the system structure, detailing LiDAR technology fundamentals (2.1), Luminar Iris sensor characteristics (2.2), and the coordinate system framework (2.3). Sect. 3 presents the theoretical background, covering range image concepts (3.1), K-D trees (3.2), object clustering fundamentals (3.3), and a review of various clustering algorithms (3.4). The methodology is then described in Sect. 4, covering data processing techniques (4.1), challenges with range image clustering (4.2), and clustering algorithms including HBAC, RGAC, KD tree merging, and HDBSCAN baseline comparison (4.3). The evaluation strategy is detailed in Sect. 5, including visual evaluation (5.1) and quantitative evaluation (5.2). Sect. 6 presents the results regarding ground removal (6.1), K-D tree merging (6.2), baseline determination (6.3), visual (6.4), quantitative (6.5), and computation time analysis (6.6). Sect. 7 contains the discussion of the results followed by the conclusion in Sect. 8.

1.7 Related Work

The clustering of LiDAR point clouds has been extensively studied in the context of autonomous driving. Several methods have been developed to deal with multi-layered data, range image projections, and problems caused by non-uniform point densities. In this section an overview of significant clustering approaches will be provided, categorizing them by the technique type, and compares them to our proposed ways.

1.7.1 Breakpoint and Distance-Based Clustering

Autonomous vehicle applications require LiDAR systems to utilize 3D sensing capabilities or multi-layer configurations to accurately capture and represent surrounding environments. A multi-layer LiDAR collects data from several horizontal layers of the laser scanner, giving a 3D view of the environment. One of the early methods to address the challenge of clustering this kind of multi-layer data was introduced by [29]. This clustering method is also explained in detail in Section 3.4.7. The thresholding technique is a distance-based clustering approach called Adaptive Breakpoint Detection (ABD), where LiDAR points are processed sequentially. The input data for the algorithm must be sequentially ordered (i.e., neighboring points are after each other in the array). For efficiency, the comparisons are limited to a small set of candidate points, which are the last point in each layer.

While this method has some similarities with the developed approach in this thesis, there are still some differences. Firstly, this approach extends ABD by adding an angle-based criterion (either a fixed threshold or a dynamic angular threshold based on Dual Break Point (DBD)), which improves separation in complex scenarios. Secondly, instead of requiring the input data to be ordered, the approaches in this thesis are based on the transformation of the 3D point clouds into a 2D-matrix like format called range image. This image inherently preserves the adjacency relationship between neighboring points by mapping them to neighboring cells. Lastly, in [29], it

is assumed that there is a uniform angular resolution in both horizontal and vertical directions, which is not ideal for the setup involving a non-uniform vertical resolution. Despite this limitation, the method avoids some of the common issues in range image clustering problems, such as gaps in the range image.

1.7.2 Range Image Clustering

A flexible class of clustering methods that can be applied in a variety of ways is called Range Image clustering. These methods efficiently handle LiDAR data points using techniques like Connected Component Labelling (CCL) [25] or region-growing [55]. A common shortcoming of all of the Range Image clustering methods as discussed in Sect. 1.7.2 is the assumption of fixed sensor angular resolution. An approach from [25] uses four-connected neighborhoods, just like in the two approaches developed in this research. However, one limitation of this work is the predefined fixed distance threshold, which causes over-segmentation for objects further away. A notable innovation in their work is the introduction of Map Connections (MCs), which explicitly connect non-neighboring points during the clustering of the range image as a solution to missing data or partial occlusion. The methods developed in this project address similar issues using K-D trees during a post-processing step, but this adds computational overhead compared to Hasecke's integrated solution.

1.7.2.1 Divide-and-Merge Strategies

Another approach explained in [55] is divided into a two-step process. First, it performs clustering locally within evenly divided regions, followed by a merging step based on edge-point pair voting. However, this algorithm is highly dependent on the choice of the starting seeds, which are initial LiDAR points selected by dividing the 3D world into voxels and picking one point from each voxel that contains points. A related divide-and-merge approach is proposed by [54]. This method performs a three-pass range image method using fixed Euclidean distance thresholds for both intra-channel (horizontal) and inter-channel (vertical) grouping. The main shortcoming of this method is its complexity, as the main focus is hardware resource optimization. In contrast, the developed approaches are simpler, intuitive and easier to implement.

1.7.3 Clustering with Non-Uniform LiDAR Data

Even though research on non-uniform LiDAR clustering techniques is still limited, several approaches have been developed to address different versions of non-uniformity, each with its own focus and strategy. In [50] there is extra emphasis on finding shapes and edges of the point cloud by breaking the data into smaller uniform sections using an adaptive clustering approach. Another related method is mentioned in [33], where point clouds are divided into smaller regions with assumed uniform density, and then DBSCAN is applied within each. In a similar fashion, the proposed HBAC method relies on a similar approach by dividing the vertical dimension into regions with roughly uniform point density in the vertical direction.

Another non-uniform approach is discussed in [22], which is designed for autonomous driving systems. The method addresses non-uniformity by dynamically adjusting the shape and size of the neighborhood used for clustering. It replaces fixed, circular neighborhood zones (as in DBSCAN) with adaptive elliptical ones. By doing so, it better handles changes in point density at different distances and directions from the sensor since the elliptical shape parameters are dynamically adjusted based on the position of the point being considered. This is quite similar to the approaches HBAC and RGAC in this thesis, since the parameters are also being changed based on the position of the considered point and its elevation. However, this approach is a density-based clustering, which is usually computationally slow in comparison to range-image methods.

2

System Structure

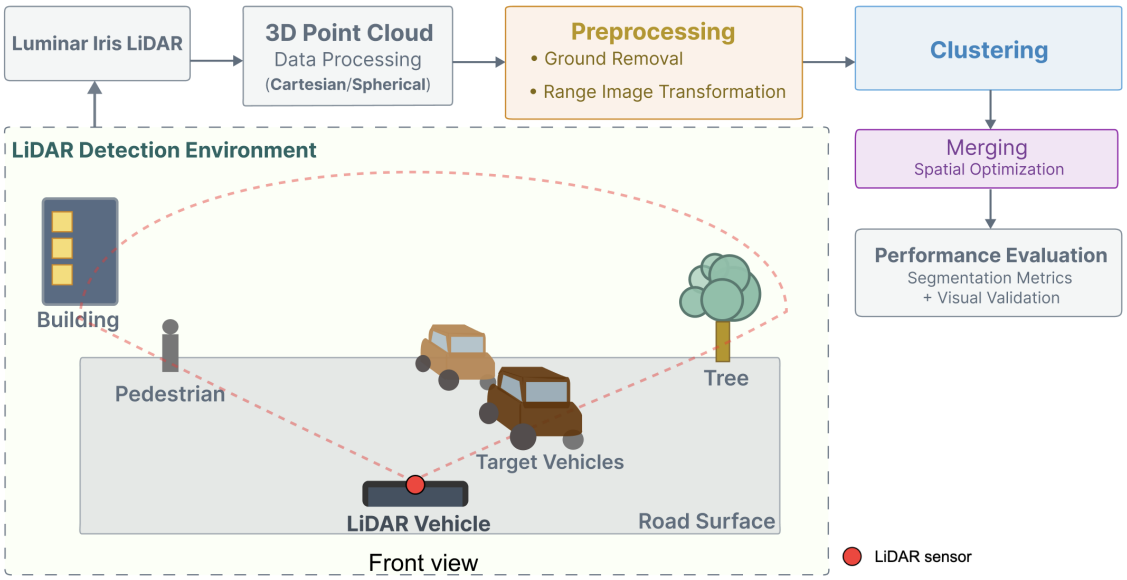


Figure 2.1: System Overview for Model-Based Clustering of LiDAR Point Clouds, processing pipeline for Luminar Iris LiDAR data; The LiDAR Detection Environment (bottom) shows typical urban scenarios requiring adaptive segmentation approaches.

In this chapter there will be a detailed explanation of the system architecture, the fundamentals of LiDAR technology, the Luminar Iris sensor and the coordinate system used for data processing and analysis. Figure 2.1 illustrates the system pipeline starting from the LiDAR point cloud, followed by a preprocessing step, clustering and a post-processing merge step.

2.1 LiDAR Operating Principle and Data Characteristics

LiDAR systems operate on the Time of Flight (ToF) principle [52]. The system first emits laser pulses in a specific direction, which are then reflected back to the sensor when they hit a surface in the environment. The system calculates the distance by measuring the time it takes from emission to reception using the speed of light as

shown in Figure 2.2. The specific formula is shown in Equation 2.1:

$$\text{Distance to target} = \frac{\text{Time of Flight} \times \text{Speed of Light}}{2}. \tag{2.1}$$

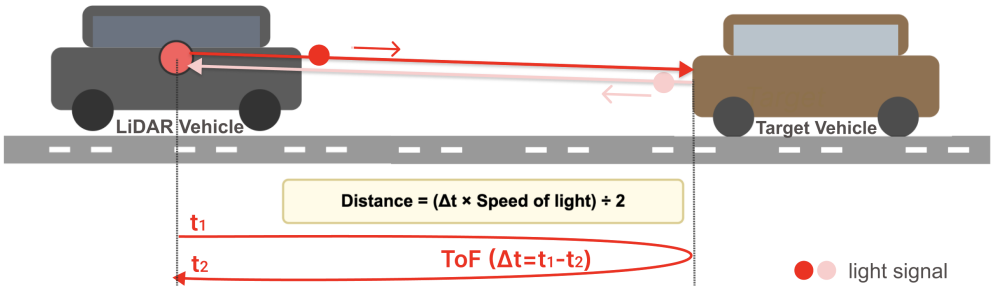


Figure 2.2: Concept of LiDAR. A light signal is emitted by the scanner and reflected off the target.

By emitting laser pulses in rapid succession at different azimuth and elevation angles, the system is able to construct an accurate three-dimensional representation of the surrounding environment [52].

Modern automotive LiDAR systems produce data in the form of a 3D point cloud, where each point represents the location of a reflected surface of an object in the environment. In the multilayer LiDAR system used in this study, the data is organized according to scan lines that correspond to different elevation angles. The points are sequentially arranged according to azimuth and elevation scanning patterns as shown in Figure 2.3, and this structured organization facilitates efficient subsequent processing and analysis [52].

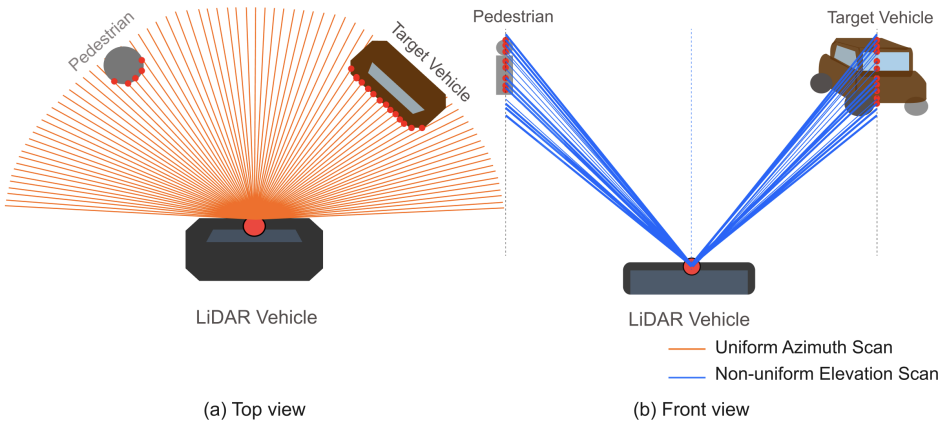


Figure 2.3: Visulaisation of LiDAR scanning pattern organization. (a) Top view shows uniform azimuth scanning. (b) Front view shows non-uniform elevation scan lines (blue) with varying angular spacing typical of automotive LiDAR systems.

2.2 Luminar Iris

The Luminar Iris used in this study is a forward-facing LiDAR system which is a vital piece of the advanced driver assistance systems (ADAS) and autonomous driving technology [35]. The biggest difference between standard LiDAR systems and Iris sensor is its non-uniform point distribution feature, which allows for higher resolution in areas of interest and reduced data in less critical regions. However, this also introduces unique challenges for point cloud processing and clustering algorithms.

2.2.1 Hardware architecture

The Luminar Iris utilizes a pulsed laser system combined with a dual-mirror scanning mechanism consisting of a polygon mirror for horizontal scanning and a scanning mirror for vertical scanning. The main hardware components of the system include the sensor head assembly (which contains the laser transmitter and receiver optics), the scanner assembly (which contains the polygon mirror and the scan mirror), the laser transceiver (XCVR) assembly, and the receiver system using an avalanche photodiode (APD) [35].

In the optical path design, the laser beam first strikes the high-speed rotating polygon mirror for horizontal scanning and then reflects back to the scanner mirror for vertical scanning. This dual-mirror design enables precise control of azimuth and elevation angles. After the laser pulse strikes the target, the light is scattered in all directions, a small fraction of which is returned to the sensor, where it is converted optically and processed for distance and other attributes. This complex optical and electronic system design enables the Luminar Iris to deliver high quality point cloud data in a wide range of environmental conditions [35].

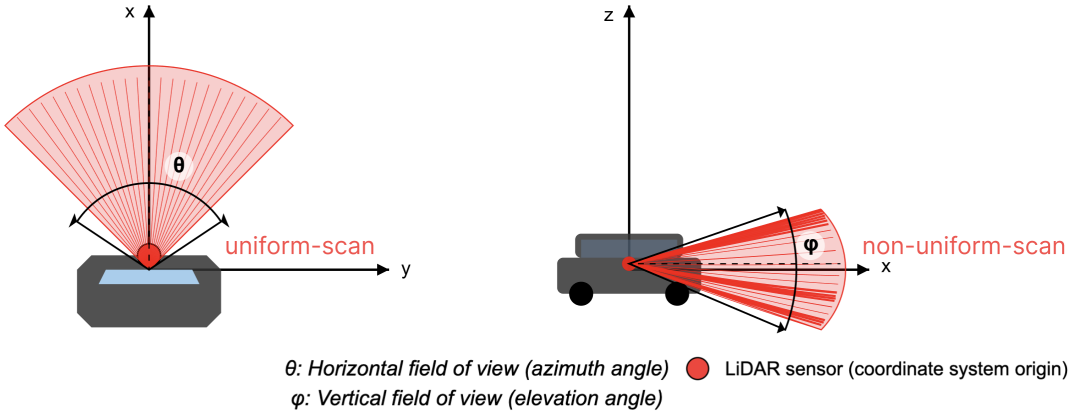
2.3 Coordinate system selection

When processing LiDAR data, the appropriate choice of coordinate system is essential for effective spatial information representation and manipulation. The Cartesian coordinate system is preferred in clustering algorithms that handles the thresholding based on Euclidean distance. The spherical coordinate system is suitable for the clustering algorithms that need thresholding based on azimuth and elevation angles, and range. In this study, a combination of coordinate systems is used, each with its own characteristics and advantages.

2.3.1 Sensor-centered Cartesian coordinate system

In this study the main coordinate system that is being used is the the Cartesian coordinate system with origin being the optical center of the LiDAR sensor. In this coordinate system, the x-axis points in the forward direction of the vehicle, the y-axis points in the lateral direction (positive left), and the z-axis points in the vertical upward direction as shown in Figure 2.4.

The main advantage of this coordinate system is its innate ability and works coherently in the vehicle motion planning. So that position, speed, and acceleration of objects can be used directly in tasks like path planning and collision avoidance mechanisms. In addition, the system makes it easier to perform common geometric and mathematical operations such as distance calculations, plane fitting and cluster analysis.



(a) Top view (x,y plane) (b) Lateral view (x,z plane)

Figure 2.4: Visualisation of Cartesian coordinate system. (a) Top view shows uniform horizontal scanning with azimuth angle θ. (b) Lateral view shows non-uniform vertical scanning with elevation angle ϕ. LiDAR sensor serves as coordinate origin.

2.3.2 Spherical Coordinate System

As mentioned in Section 2.3.1, Cartesian coordinate is powerful for many downstream clustering tasks. However, the LiDAR sensor samples data in the spherical coordinate system. This makes interpretation and processing of the raw data straightforward.

In the spherical coordinate system, each point is represented by three parameters r, θ and ϕ. The distance (r), represents the euclidean distance from the sensor to the measurement point. The Azimuth (θ) represents the angle measured from the x-axis in the horizontal plane while the elevation (ϕ) highlights the vertical angle measured from the horizontal plane.

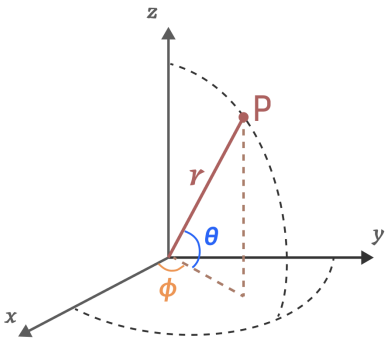


Figure 2.5: Visualization of a Spherical coordinate system.

The conversion from spherical to Cartesian coordinates can be achieved by using the following equations:

$$\begin{aligned} x &= r\cos(\theta)\cos(\phi) \\ y &= r\cos(\theta)\sin(\phi) \\ z &= r\sin(\theta) \end{aligned} \tag{2.2}$$

Conversely, Cartesian coordinates can be converted to Spherical coordinates using the following equations:

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= \arctan(y/x) \\ \phi &= \arcsin(z/r) \end{aligned} \tag{2.3}$$

3

Theory

In order to comprehend the methodology and results of this thesis, in this chapter a thorough description of different clustering techniques and algorithms, along with their drawbacks and advantages is presented.

3.1 Range Image

Regular image gives information about color and brightness which are important features for understanding how the entire scene looks like [48]. On the contrary, a range image provides information about the distance from the sensor to different objects in the environment. The main difference between two types of images is that instead of colors, each pixel in the range image stores a range value. The rows represent the vertical laser beams, and the columns represent the horizontal rotation steps.

In addition, range images are useful in industrial applications, where it checks the 3D shape of parts to spot defects and make sure everything meets dimension requirements. They are also useful in robotics, where knowing the distance and shape of objects is important when picking up or moving items around safely. A robot needs to know where the object, how far away, and its shape to grab it [48].

Batchelor and Waltz delve into other techniques, such as edge detection and region segmentation, to extract more important information from range images. Edge detection is a technology that helps find boundaries between surfaces, while region segmentation groups pixels together that belong to the same object [48].

3.2 K-D Tree

K-D tree is commonly used to organise spatial data because of the efficient operations it has, such as nearest neighbor searches and range queries which are vital to LiDAR point cloud processing. A K-D tree is essentially a binary tree with internal nodes, where each node represents a K-dimensional point [39]. The process of constructing a K-D tree begins by choosing an axis and selecting the median point along the axis to split the tree into two subsets. Following that, a child of the root node is created and placed on either the left or right side, depending on its relation to the

median. To ensure efficient query (i.e., search) operations, the tree is kept balanced by a continuous recursive splitting across different axes.

The query step is about using this structure to efficiently traverse the tree when finding the closest neighbor to a given point. This tree data structure is great alternative since the data is already organised spatially and many branches can be skipped during the query.

The average-case complexity for the insertion and the query steps is O(log n). However, in the worst case, when the tree is unbalanced, the complexity becomes O(n). Conversely, the tree creation has average-case complexity of O(n log n) [39] where n is the total number of K-dimensional data points that are inserted into the K-D tree.

3.3 Object Clustering

In autonomous driving technology, clustering of LiDAR data is crucial to understand the surrounding environment [24]. Object clustering is the action of grouping similar data points based on certain features such as reflectivity and euclidean distance [38]. This technique is often used in applications such as image segmentation, pattern recognition, and data analysis. Examples of clusters are vehicles, pedestrians or road signs. Moreover, clustering significantly improves the efficiency of downstream tasks in autonomous driving, such as object tracking and detection [3].

Various clustering algorithms can be used depending on the data and the objective. To find object boundaries for instance, some approaches interpret the point cloud as unordered and use only spatial distance, whereas other way treat it as ordered and use the sequence of collected points [8]. Typically, the objective is to balance speed and high accuracy as autonomous systems must make decisions in real time [32].

3.3.1 Over-Segmentation and Under-Segmentation

Over-segmentation is when an object, such as a car, is interpreted by a clustering algorithm as more than one object, being split into pieces [27]. This issue can lead to poor tracking of the object in question.

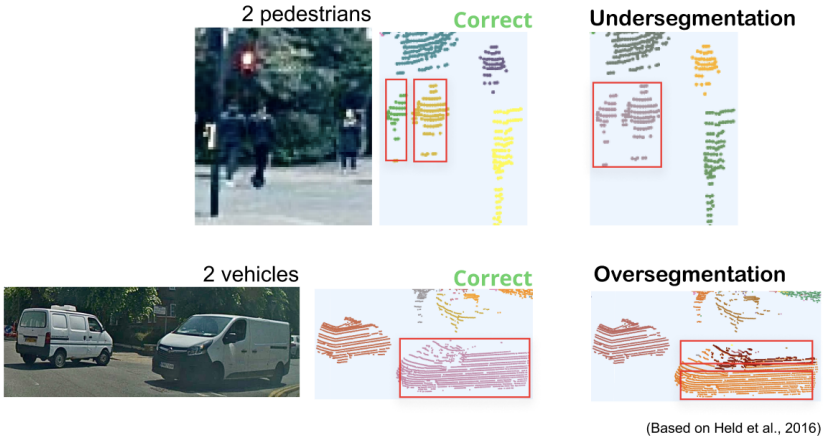


Figure 3.1: Illustration of Over- and Under-segmentation in a LiDAR Scan, inspired by [27].

Under-segmentation is the opposite, when multiple objects are being interpreted as one object. Examples of this could be a pedestrian being interpreted as part of a tree or a light pole, which could lead to serious accidents, due to the algorithm not detecting the pedestrian in time to react [27]. If a pedestrian is misclassified as part of a tree, the system may ignore the object altogether, assuming it's static or irrelevant, leading to no response when one is critically needed.

3.4 Types of Clustering Algorithms

In this section, a discussion over different clustering algorithms, including partition-based clustering, density-based clustering, hierarchical clustering, deep learning methods, and techniques specialized for clustering, such as breakpoint detection. The aim is to offer a deep dive into each method and provide understanding when to apply which clustering method.

3.4.1 Partition Clustering

Partition clustering is an iterative approach and is based on the similarities of points and their distance to the center of each cluster [3]. An example of partition clustering is an unsupervised learning approach called K-Means. This technique works by dividing the dataset into k different spherical clusters.

The first step of K-means algorithm is selecting k random initial centroids or seeds. Thereafter, each data point assigned to the cluster with the nearest (i.e., in terms of distance) mean (centroid) [49]. The centroids must be recalculated once all points have been assigned to each initial centroid by taking the mean of all points within each cluster. The process of centroid calculation and point assignment continues iteratively until a maximum number of allowed iterations is reached or when the cluster assignment no longer changes significantly [49].

The main advantages of K-Means method are its speed and simplicity, which make

it great for large datasets [16]. Yet some limitations exist. The algorithm is sensitive to the first choice of the centroids and can converge to a local optimum instead of the global optimum. It also assumes that the clusters are equally sized and spherical, which is usually not true for real-world data [46].

The inability of K-Means to handle noise or outliers effectively is a flaw that can skew the final clustering result. Even a few extreme numbers can drive clusters in an incorrect direction, which distorts the final results, because the algorithm uses data point averages to determine cluster centers [19]. Consequently, K-Means is less optimal choice when working with complex data structures such as non-uniform LiDAR point clouds. Another disadvantage of K-Means is that it is sensitive to stray points such as rain, dust or any other outliers as they can be conflicted with real objects points. However, the performance of K-Means is compared to the results from more complex clustering strategies.

3.4.2 Density-Based Clustering

Density-based clustering is a technique used to identify groups of data points that are tightly packed together, while treating those located in sparse regions as outliers [20]. A popular method within this category of clustering is "Density-Based Spatial Clustering of Applications with Noise", shortened to DBSCAN. It's especially useful for working with datasets that contain irregularities.

DBSCAN does not assume that clusters are spherical nor that the number of clusters be predetermined, which K-Means [7]. It is instead based on two key parameters: *minPts*, which specifies the minimum number of nearby points needed to construct a dense zone, and ε (epsilon), which specifies the maximum distance between two points to be considered neighbors [6]. If, for a point, there are at least *minPts* neighbors within a distance ε, that point is deemed a core point. The same cluster is expanded by the method through recursively inspecting neighbors of neighbors, and points that are in the vicinity of a core point are added to the cluster.

However, DBSCAN also has limitations. The way ε and *minPts* are chosen can have a significant impact on the algorithm's performance [45]. A large value of ε could lead to an incorrect merging of different clusters, while a small value could result in too many points being classified as noise. A single ε value might not be appropriate across different spatial regions, temporal conditions, or data distributions; therefore, DBSCAN may have trouble with datasets with different densities [17].

A modified version of DBSCAN called Hierarchical DBSCAN (HDBSCAN) solves this issue. HDBSCAN creates a hierarchical tree of clusters by gradually changing the density threshold [13]. As the density requirement becomes stricter, clusters keep changing, and then the most stable clusters are selected, meaning those that persist across a wide range of density levels. This allows HDBSCAN to better handle clusters of varying densities while still identifying noise effectively.

When used on large datasets, DBSCAN can also have a high computational complexity. The algorithm's worst time complexity can reach $O(n^2)$ if effective spatial indexing

structures such as K-D trees are not used [20], [9]. This is a problem for real-time processing applications like robotics or autonomous driving [12].

3.4.3 Hierachical Clustering

Hierarchical clustering is an unsupervised machine learning method that builds a hierarchy of clusters by either following an agglomerative approach, where smaller clusters are merged into larger ones, or following a divisive approach, where larger clusters are split into smaller ones [37].

The primary benefit of Hierarchical clustering is that the technique does not require a predetermined number of clusters, which makes it particularly useful for exploratory data analysis where the dataset's underlying structure is unknown i.e., when analysts lack prior knowledge about natural group boundaries or hierarchical relationships within the data [21], [51]. Because of its adaptability, the algorithm can naturally identify patterns and linkages. Furthermore, thanks to the results usually being displayed in a tree-like diagram that demonstrates nested grouping of data points and the order in which the clusters are merged or divided, hierarchical clustering is easily interpretable and intuitive [26].

There are drawbacks to hierarchical clustering, particularly for large datasets, it is computationally more costly than partition-based techniques like K-Means since it needs to compute and store a whole distance matrix [37]. Also, once a merge or split is done, it cannot be reversed, which can result in worse clustering choices [51].

There are different linkage criteria used to measure the distance between clusters during the merging process, e.g., single link or complete link [28]. The shortest distance between any two points from each cluster is the single-link definition of the distance between two clusters. Despite its computational efficiency and ability to capture non-elliptical patterns, hierarchical clustering frequently suffers from the "chaining effect", which is the tendency to link points that may be loosely connected by a series of close neighbors to form elongated and incorrect clusters [2]. In certain datasets, this may result in clusters that are excessively stretched. Complete link, on the other hand defines the inter-cluster distance by taking the maximum distance between any two sites in distinct clusters [28], [31].

3.4.4 Grid-based Clustering

An appropriate method to deal with vast datasets is grid-based clustering, which summarizes the datasets with a grid representation, and afterwards merges grid cells into clusters [4]. This is to avoid working with specific data points, instead quantizing the data space with a grid to create a more geometrical outline of the data. The clusters are created by accumulating adjacent heaped cells. Grid-based clustering can be likened to density-based clustering, but focuses on local densities and neighborhood relations between the cells without regarding the relation between individual points [4]. Grid-based clustering can therefore offer advantages in efficiency by reducing problem space, improving scalability by having the capacity to handle

multi-dimensional data, and flexibility due to its adaptation to different distributions by using clustering techniques.

3.4.5 Deep Learning methods

Machine learning and neural networks are essential techniques for object recognition, and the automotive industry currently relies heavily on them [36], [41], [42]. There are multiple ways to use LiDAR sensors and deep learning to detect 3D objects around self-driving cars [53]. Since LiDAR data is unordered, these methods typically convert the data into simpler, organized shapes like blocks (voxels) or groups of points (pillars). After the data is organised, it can be input into several deep learning models to identify and classify objects like cars, pedestrians, or cyclists. Additionally, deep learning is used in LiDAR-camera fusion, where models are trained on both images and point clouds to improve object detection accuracy [23].

However, these methods have limitations in real-world applications, This method also requires large high quality labeled datasets for training [18]. To get a labeled dataset, the annotation procedure is used, which is expensive and resource-intensive for 3D point cloud data, either requiring advanced tools or human labeling [3]. In contrast, classical clustering algorithms have advantages when compared to deep learning methods. Firstly, they do not rely on large-scale labeled datasets for training, they can be more computationally efficient, and their behaviors are more predictable and interpretable, properties that are particularly important for safety-critical applications such as autonomous driving [51].

One of the major drawbacks of deep learning methods is its inherent black-box nature because the internal decision-making processes of these models is not clear and intuitive. This makes it difficult to interpret and analyse the specific reason of the output [47], [15]. This lack of interpretability poses significant challenges in autonomous driving, where trust, transparency and the ability to identify failure modes are necessary for ensuring safety [22].

3.4.6 Breakpoint Detection

Breakpoint detection involves dividing raw sensor data into clear clusters. Ideally, each cluster will correspond to a different surface or item in the environment [5].

The Euclidean distance between successive scan points is the most straightforward and widely used breakpoint detection approach [29]. As the LiDAR beams scan over the surroundings, the LiDAR sequentially takes measurements [5]. Two consecutive points, such as p_i and p_{i+1}, belong to the same continuous surface or object if they are near enough to one another, meaning that they do not exceed a distance threshold D_m

$$\|p_i - p_{i-1}\| > D_{\mathrm{m}}. \tag{3.1}$$

On the other hand, if the distance between them is above D_m, it is an indication that there is a breakpoint between p_i and p_{i+1}. Consequently, p_{i+1} would then be

counted as the first point of a new cluster in that scenario, whereas p_i would be the last point of one cluster.

One challenge with a fixed distance threshold is that the density of the acquired points decreases as the distance from the sensor increases. Therefore, a fixed distance threshold may cause under-segmentation for the furthest objects and over-segmentation for closer objects.

3.4.7 Adaptive Breakpoint Detection

One potential solution to the"the previously mentioned challenge is adjusting the distance threshold based on the distance to the LiDAR sensor. An example of such a method is Adaptive Breakpoint Detection (ABD) [11]. The key concept of the ABD is an imaginary line as presented in Figure 3.2. This line connects a hypothetical point p_h^n and the previous point p_{n-1} at a range r_{n-1} and angle φ_{n-1} from the LiDAR, making a small user-defined angle λ, called worst-case incidence angle.

The incidence angle refers to the angle at which the sensor's emitted signal hits the surface of an object in the environment. In other words, it represents the surface orientation relative to the direction of the beam measurement from the sensor. This angle is interpreted as the worst case incidence angle where points can be detected [11].

For instance, the sensor's beam touches the surface almost parallel to it if the incidence angle is close to 0°, which results in a sparser LiDAR points on that surface. On the contrary, if the incidence angle is almost 90° (i.e., sensor is almost directly facing the surface) the LiDAR points will be more densely packed. This is illustrated in Figure 3.3 where the side with the larger incidence angle has tighter points than the side with the smaller incidence angle [11].

To estimate the maximum allowed distance between consecutive points on the same continuous surface at range r_{n-1}, a hypothetical next point p_h^n is considered. This maximum distance, $||p_h^n - p_{n-1}||$, is determined using the sensor's angular step $\Delta\varphi$ and basic trigonometric relationships based on the triangle formed by p_{n-1}, p_h^n, and the sensor's position. The equation is shown in Equation 3.2

$$||p_h^n - p_{n-1}|| = r_{n-1}\frac{\sin(\Delta\varphi)}{\sin(\lambda - \Delta\varphi)}. \tag{3.2}$$

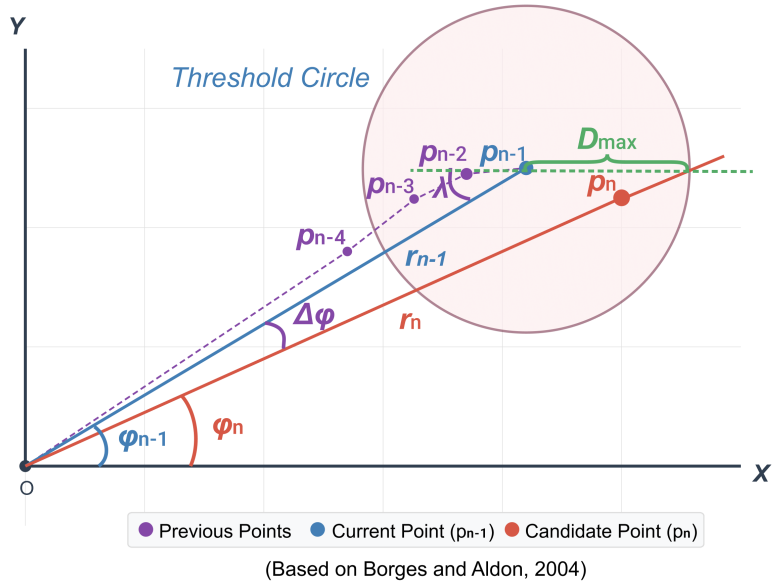


Figure 3.2: An adaptive threshold detection method. p_{n-2}, p_{n-1} and p_n are adjacent points; a dotted line (shown in green) is drawn from point p_{n-1}, forming an angle λ with the blue scanning direction r_{n-1}. This extrapolation line is used to predict the position of range point p_n. This figure is derived from the work of [11].

Lastly, [11] suggests increasing this hypothetical distance by a margin that accounts for sensor measurement noise. Adding a multiple of the sensor's range resolution σ_r, usually $3\sigma_r$. Consequently, the adaptive distance threshold D_{max} is obtained as shown in equation (3.3)

$$D_{max} = r_{n-1}\frac{\sin(\Delta\varphi)}{\sin(\lambda - \Delta\varphi)} + 3\sigma_r. \tag{3.3}$$

3.4.8 Parameter Selection for ABD

The choice of the mentioned angle λ is vital for the results of the ABD method.

- A value of λ that is too high will lead to over-segmentation (i.e., points from the same object are not clustered together) [11].
- A value of λ that is too low, on the other hand, could result in under-segmentation by combining points from different objects [11].
- The optimal value for λ is usually found empirically by considering the specifications of the sensor and the environment being scanned. In certain situations, a value of $\lambda = 10°$ has been reported as satisfactory [11].

3.4.9 Dual Breakpoint Detection

Dual Breakpoint Detection (DBD) further builds on ABD by adding a criterion based on the angle between consecutive scan points or their incidence angles concerning

the sensor [5], [40]. There are some cases where ABD misclusters two points from the same surface. However, an additional angular criterion can detect and correct these misclusterings.

In particular, the collinearity of three consecutive LiDAR points p_{i-2}, p_{i-1}, and p_i is investigated. If these three points lie approximately on a straight line, even if the distance between p_{i-1} and p_i is relatively large, they might still belong to the same cluster, as can be seen in Figure 3.3. In DBD, the distance-based criterion is usually checked first, often employing an adaptive threshold. If the distance criterion is not met, especially if the current cluster contains only one or two points, the angle-based criterion is then evaluated [5].

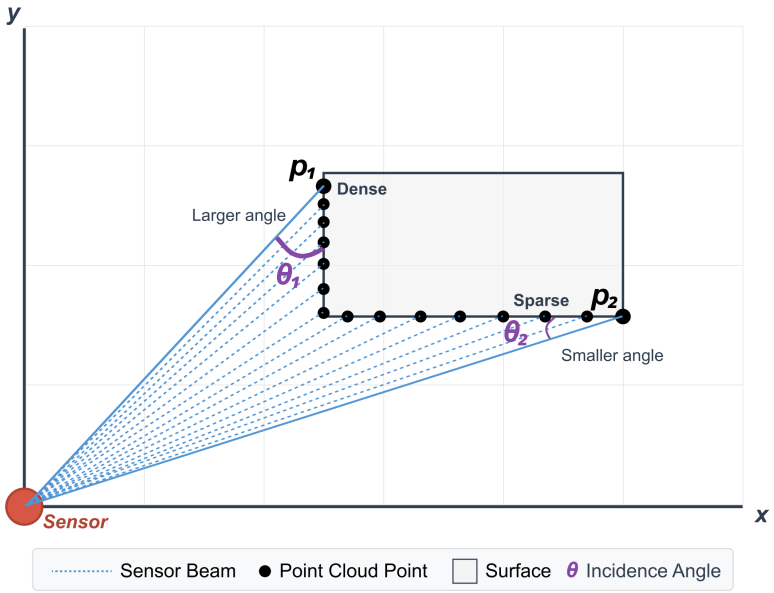


Figure 3.3: Two Incidence angles θ_1 and θ_2 where the points are denser on the surface with the larger incidence angle θ_1. This is derived from the work of [40].

Consider the angles β_i and β_{i-1}, formed by the pairs of successive points (p_i, p_{i-1}) and (p_{i-1}, p_{i-2}), respectively. If the difference between the two angles is below a certain angle threshold α_{DBD}, it suggests that the points are part of a continuous linear structure. This is represented in the following equation

$$|\beta_i - \beta_{i-1}| < \alpha_{DBD}. \tag{3.4}$$

According to Johansson & Wellenstam [40], and Su-Yong et al. [5], clusters formed using DBD from LiDAR point clouds are categorized into two distinct types based on the criteria used to group their points: distance-based clusters (DC) and angle-based clusters (AC) [40], [5]. In essence, a cluster is labeled as DC if it begins with at least one point satisfying the distance threshold. From that point onward, only the distance criterion is used to expand the cluster [40], [5].

If no point satisfies the distance threshold at the start, the algorithm instead applies

the angle criterion. In this case, the cluster becomes an AC, and new points are added only if they meet the angular continuity condition. However, a new cluster is started if the subsequent point does not fulfill the respective criterion once the cluster is already labeled AC or DC. This dual labeling system gives the distance criterion priority when it applies, and the angle criterion when distance is insufficient [40], [5].

3.4.10 Range Image Clustering

A clustering method that has gained popularity recently is range image-based clustering. This approach performs clustering on a 2D range image instead of working directly on a 3D point cloud, which lowers computational cost significantly while preserving distance relationships between points [10].

In a range image, each pixel or cell represents a range value measured by the LiDAR, which effectively captures the 3D structure in a 2D grid format [10]. This structure means that neighboring cells correspond to neighboring LiDAR beam measurements, making it easier to identify adjacent points using fixed 2D indices such as left, right, up, and down. In contrast, the 3D point clouds are scattered and distributed in space without any inherent clear order. This makes finding nearest neighbors computationally expensive, especially in denser regions, since a lot of calculations are necessary. However, this burden is removed naturally by the regular structure of a range image, which enables much faster and more reliable clustering [10].

This clustering approach determines whether two neighboring points belong to the same physical object [10]. The decision is made by calculating an angle β. Based on Figure 3.4, suppose that the laser scanner is at point O, and it measures two points A and B in space using adjacent laser beams. The angle β is defined at the furthest of the two points, for example point A, as the angle between the vector from O to A and the line connecting A to B.

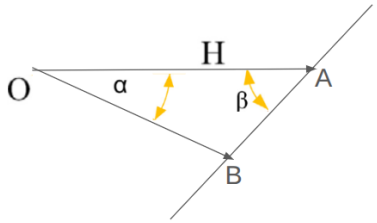


Figure 3.4: Geometric relationship of neighborhood points based on angle β.

This angle serves as a simple yet effective test of surface continuity. If two points lie on the same smooth surface, the angle β between them will be relatively large, since their range values will be similar despite the angular separation between the laser beams. However, if there is a sharp range change, e.g., the laser beams hitting different objects or parts of an object with sharp edges, the value of β becomes smaller. Consequently, a small β indicates a likely boundary between different objects.

The angle β can be computed using basic trigonometry. Given two consecutive range measurements d_1 and d_2, and the known angle α between their respective laser

beams, β is calculated :

$$\beta = \arctan\left(\frac{d_2 \cdot \sin(\alpha)}{d_1 - d_2 \cdot \cos(\alpha)}\right). \tag{3.5}$$

Moreover, the clustering process now becomes a connected-component labeling task on the range image [10]. Using a standard four-connected neighborhood, the algorithm steps through the image, starting from each unvisited cell. It checks adjacent cells to determine whether they belong to the same object, based on how smoothly they connect. This connection is tested using the angle β and a predefined threshold θ. The predefined threshold θ angle defines the minimum β value required for two points to be grouped. It controls how tolerant the algorithm is to range variations between neighboring points. If $\beta < \theta$, meaning the difference in range is too large, it will imply surface discontinuity and thus the points are not grouped. If $\beta \geq \theta$, the range change is acceptable, and the points are considered part of the same object. This process continues recursively, grouping all cells that are connected. In doing so, the algorithm efficiently identifies distinct clusters while ensuring a cell is visited only if necessary [10].

However, a limitation of this method is that this approach assumes that the sensor has the same angular resolution in both the vertical and horizontal directions [10]. This assumption does not hold for the LiDAR used in this study, which has a non-uniform distribution of beams in the vertical direction. Therefore, the goal of this thesis is to build on the strengths of this methodology and adapt it to handle non-uniform angular resolutions. The aforementioned algorithm will be referred to as the baseline approach in the upcoming chapters.

4

Methods

In this chapter, the entire methodology of the two different clustering approaches is explained. The data processing steps, different ground removal techniques, and range image creation are covered in detail. Subsequently, a comprehensive explanation of the challenges in range image-based clustering will be provided. Thereafter, the design of the two approaches HBAC and RGAC will be explained in depth with justification of respective design choices. Finally, the chapter concludes with a description of a post-processing step that merges individual points using K-D trees. The methodology follows a systematic three-stage pipeline as illustrated in Figure 4.1, progressing from data processing through core clustering implementation to spatial optimization via K-D tree merging techniques.

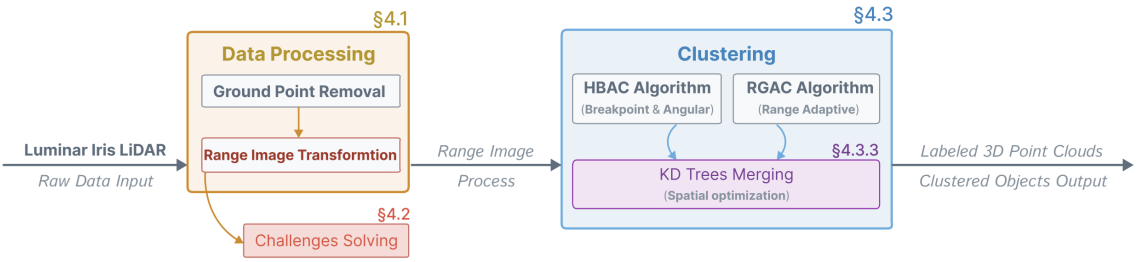


Figure 4.1: Methods flow-chart overview.

4.1 Data Processing

In LiDAR point-cloud processing, effective data preparation is crucial for accurate and efficient clustering. Our data processing pipeline consists of two key stages: (1) ground point identification and separation to isolate non-ground objects and (2) range image transformation to establish a structured representation for clustering. This section details these preprocessing steps that form the foundation of our segmentation approach.

4.1.1 Ground Point Removal

Ground points usually cover a significant portion of LiDAR data. Removing these ground points can greatly improve the performance and accuracy of subsequent clustering algorithms, as they prevent false connections between spatially separated

objects. This section details our coordinate transformation-based approach to detect and remove ground points using sensor orientation information.

4.1.1.1 Algorithmic Implementation

The ground point removal algorithm operates within the coordinate system framework established in Section 2.3. As described previously, our processing pipeline utilizes three coordinate systems: the raw LiDAR data in spherical coordinates, the sensor-centered Cartesian system for computational processing, and the calibration reference system centered at the vehicle's rear axle. The coordinate transformations between these systems, detailed in Section 2.3, enable accurate ground point identification by leveraging sensor orientation information expressed as quaternions relative to the calibration reference frame.

The transformation process involves two key steps to ensure accurate ground point identification. First, a quaternion-based rotation is applied. Quaternions ($q = w + xi + yj + zk$) provide an efficient and stable way to represent 3D rotations compared to Euler Angles[30]. The quaternion is converted to a rotation matrix using the formula:

$$R = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{bmatrix} \tag{4.1}$$

where w, x, y, and z are the normalized quaternion components such that $w^2 + x^2 + y^2 + z^2 = 1$.

The second step involves converting the LiDAR Coordinate System to the Calibration Reference Coordinate System through a combination of rotation and translation:

$$\mathbf{P}_{ref} = R \cdot \mathbf{P}_{lidar} + \mathbf{t} \tag{4.2}$$

where $\mathbf{P}_{ref}$ and $\mathbf{P}_{lidar}$ represent the point coordinates in the reference and LiDAR coordinate systems, respectively; R is the rotation matrix from the previous step, and $\mathbf{t}$ is the position vector of the sensor in the Calibration Reference Coordinate System.

4.1.1.2 Ground Point Identification Method

The development of ground point removal follows a progressive methodology, evolving from a basic threshold technique "Fixed Threshold Method" to an advanced adaptive approach "Dual-Grid Adaptive Method" that overcomes the initial method's constraints.

Fixed Threshold Method After transforming the point cloud to the Calibration Reference Coordinate System, the initial approach employs a straightforward height-based threshold for ground point identification. This method utilizes the z-axis

alignment with the vehicle's vertical orientation to apply a fixed height threshold of -0.27 meters, corresponding to the vehicle's tire height, as detailed in Algorithm 1

Algorithm 1: Fixed Threshold Ground Removal

Point cloud P with points (x, y, z)
Ground threshold $T_{ground} = -0.27$m
Output: Boolean mask of non-ground points
Initialize mask $M \leftarrow$ false
foreach *point* $p_i \in P$ **do**
    $M[i] \leftarrow (p_i.z > T_{ground})$
return M

This approach assumes relatively flat terrain and may struggle with uneven surfaces or varying ground elevations.

Dual-Grid Adaptive Method For complex terrain, a dual-grid adaptive method is applied to handle varying ground elevations and object boundaries more effectively, by adapting to terrain characteristics using two complementary grid resolutions. The complete algorithm is presented in Algorithm 2:

Algorithm 2: Dual-Grid Adaptive Ground Removal

Point cloud P with points (x, y, z)
Parameters:
    G_L: Large grid size
    G_S: Small grid size
    T_h: Height threshold
    T_d: Maximum height difference
Output: Grid dictionaries and object areas
Begin:
Calculate point cloud boundaries: x_{min}, y_{min}
Initialize grid dictionaries D_L, D_S
// For storing height statistics **foreach** *point* $p_i \in P$ **do**
    Calculate grid indices for both resolutions
    Update min/max z-values in corresponding grid cells
Initialize dynamic object areas set $A \leftarrow \emptyset$
foreach *grid cell* idx_S *in* D_S **do**
    if $D_S[idx_S].z_{max} - D_S[idx_S].z_{min} > T_d$ **then**
        Add idx_S to A
        // Cell contains potential object

The dual-grid approach partitions the point cloud into two grid scales - a coarse grid for general terrain modeling and a fine grid for detailed analysis around dynamic objects. The algorithm identifies areas with significant height variations (potential objects) and applies stricter thresholds in these regions, preserving object boundaries while effectively removing ground points.

Algorithm 2: Dual-Grid Adaptive Ground Removal (continued)

Point cloud P, dynamic object areas A, parameters T_h
Output: Boolean mask of non-ground points

Continue:
Initialize ground mask $G \leftarrow$ false
foreach *point* $p_i \in P$ **do**
    if p_i *in dynamic object area* **then**
        $G[i] \leftarrow (p_i.z \leq \min\_z + T_h/2)$ // Stricter threshold
    else
        $G[i] \leftarrow (p_i.z \leq \min\_z + T_h)$ // Standard threshold
return $\neg G$

The quality of ground segmentation influences downstream tasks. The threshold selection involves a trade-off: aggressive ground removal may inadvertently remove parts of low-height objects, while conservative removal may retain ground points that interfere with accurate object detection. The dual-grid approach balances these considerations.

4.1.2 Range Image Transformation

After ground removal, the next critical step in the data processing pipeline is creating a range image structure, which maps the point cloud information into a range image-inspired data structure. Each entry in the range image stores a set of attributes associated with a LiDAR point, as summarized in the table shown in the top-right corner of Figure 4.2. The attributes include the azimuth, elevation, range, vertical index, horizontal index, and validity flag. To construct the 3D coordinates (X, Y, Z) of each point, it is necessary to convert the spherical coordinates into cartesian coordinates while accounting for the sensor's position within the 3D map. The transformation from spherical to cartesian coordinates is given by 2.2 where r is the measured range, θ is the elevation angle, and ϕ is the azimuth angle. The validity flag is a binary label indicating if the current point is a missing or true measurement.

The range image is first created as a grid with empty cells, shown as red squares in Figure 4.2. The size of the grid is based on the characteristics of the Luminar sensor, which has a certain maximum number of points it can detect vertically and horizontally. The structure of the range image changes depending on the scenario or environment, where some scenes result in more detections, resulting in a more densely filled image, while others produce fewer points and result in more empty cells. As the LiDAR sensor collects data, the empty cells get populated with Point Object, shown as green squares in the Figure 4.2.

The LiDAR point cloud is mapped to the range image, where the vertical axis represents each laser beam's elevation index and the horizontal axis represents the azimuth index. The elevation index is set to correspond to each point's scan line index, which is the laser beam number or index in the vertical direction from which

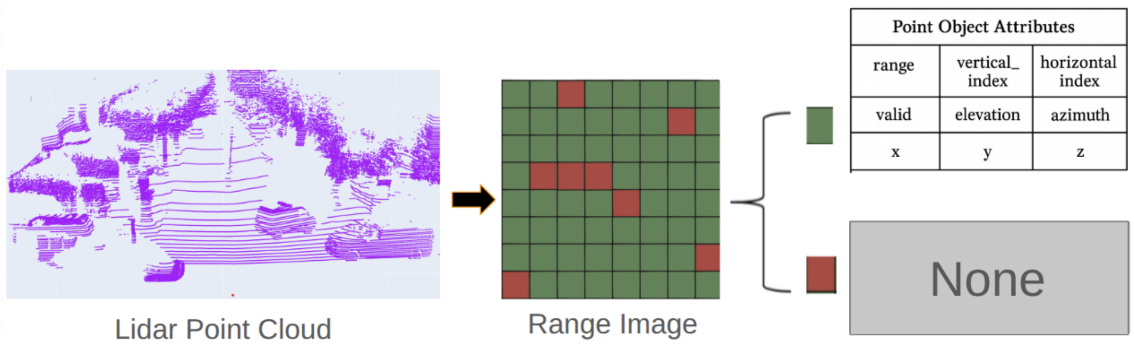


Figure 4.2: Illustration of the range image creation process. The leftmost picture is a snapshot of the point cloud for a real scenario. In the middle picture, each cell represents a point from the LiDAR scan, organized by elevation (vertical axis) and azimuth (horizontal axis). Red cells indicate missing data (no detection), while green cells contain valid LiDAR points. The top-right table shows the attributes stored in each cell, including range, azimuth, elevation, and validity.

the current point was obtained. Each index represents a distinct elevation angle in the scanning pattern of the sensor.

Meanwhile, to calculate the azimuth index, each point's azimuth angle is adjusted to fit within the sensor's field of view. This change accounts for any offset in the sensor's coordinate system, given that the field of view is symmetric, ranging from $-\theta_{\text{max}}$ to θ_{max}. However, since the range image's horizontal index must begin at zero, θ_{max} is added to compensate.

Afterwards, the angle θ_{az} is normalized by dividing it by the angular resolution of the sensor, which defines the uniform horizontal spacing between laser beams. This normalization converts the azimuth angle into a discrete column index and puts each point into the appropriate cell of the range image. This procedure ensures that the majority of the neighboring points in the 3D scan remain adjacent in the 2D range image data structure representation, thus preserving positional relationships.

The azimuth index col for a given point in the range image is computed as

$$\text{Col}_{\text{index}} = \left\lfloor \frac{\theta_{\text{az}} + \theta_{\text{max}}}{\Delta\theta_{\text{az}}} \right\rfloor, \tag{4.3}$$

where θ_{az} is the measured azimuth angle of the point, θ_{max} is an angular offset used to align the sensor's coordinate system, and $\Delta\theta_{\text{az}}$ is the angular resolution of the sensor. The floor operation $\lfloor \cdot \rfloor$ ensures the resulting index is rounded down to the nearest integer, although this step may be adjusted depending on the desired rounding method.

Figure 4.3 illustrates the range image mapping process. Both the uniform distribution and the non-uniform distribution are mapped in the same way, since the rows are mapped using the scan line index rather than the elevation. Consequently, the positional relationships were preserved. Using this conversion method of the point

cloud to a range image data structure, it becomes possible to differentiate the vertical and the horizontal neighbors by different cluster criteria in both directions.

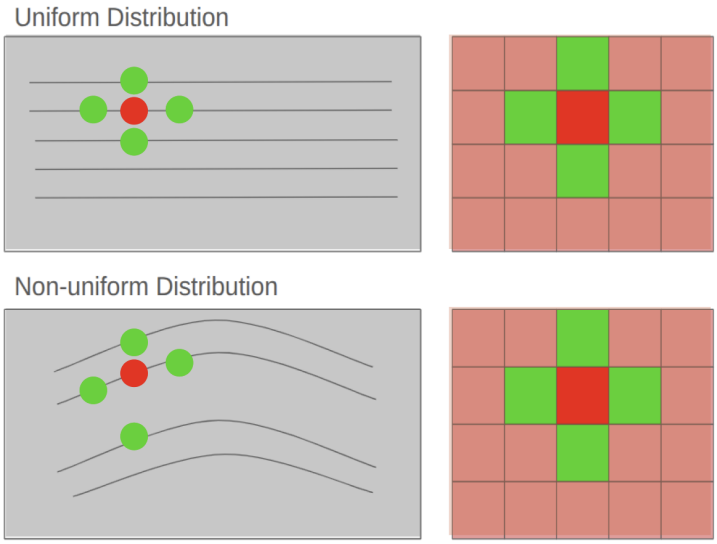


Figure 4.3: Example of range image transformation for both uniform and non-uniform point cloud distributions. The non-uniform case is for illustrative purposes and not from this project. The right subfigures show the range image, where crosses indicate detections, red for the current point, green for its neighbors.

4.2 Challenges With Range Image Clustering

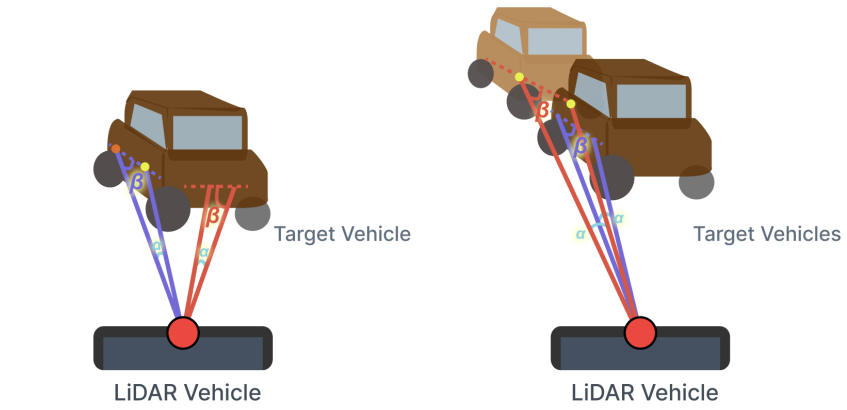
This section highlights two critical issues that have shaped the methodological choices. The first is the problem with reliance on the β angle as the sole clustering criterion, as seen in equation (3.5). The second is the impact of missing measurements, which breaks neighborhood continuity in the range image. Each of these problems is explored in detail in the following subsections.

4.2.1 Angle-Based Clustering Limitation

The baseline method [10] mainly works by comparing the angle between neighboring points. If the angle between them is big enough, the algorithm assumes they belong to the same cluster. Otherwise, they are considered part of different clusters.

This simple angle check can lead to over-segmentation or under-segmentation. This problem arises when the LiDAR is scanning flat surfaces, like a wall or a car from the side, that are sitting almost parallel to the laser beam. This problem is visualized in Figure 4.4a, where two points captured from the front surface of a car are clustered together since they have a large β angle which exceeds the $\theta_{baseline}$ threshold. In contrast, the two points from the side surface have a β that is lower than $\theta_{baseline}$, and consequently, they get clustered differently, even though they are both from the same car.

Reducing the angle threshold can help to merge such points, but it introduces a new issue. For example, if multiple cars are parked closely behind each other, points from respective cars may end up being grouped into one cluster, causing under-segmentation as can be seen in Figure 4.4b.



(a) Over-segmentation due to small angles on the longer side surface.

(b) Under-segmentation caused by a reduced angle threshold.

Figure 4.4: Clustering challenges in angle-based methods.

4.2.2 Missing Measurements in the Range Image

A problem that LiDAR sensors can be affected by is that they can produce missing connections between points in their scans. This usually happens when the laser beam doesn't return to the sensor, which can be caused by several factors. The material of the object hugely affects how LiDAR performs. For example, transparent materials like glass might enable the beam to pass through without reflection, resulting in no return signal and missing data points; Reflective surfaces like mirrors can cause multiple reflections, resulting in ghost pints that appear in the incorrect places due to the longer beam travel time. Environmental conditions like fog, rain, or dust can interfere with the laser as well [1].

Figure 4.5 shows a case where a car windshield causes missing measurements. Three blue detections are isolated from the green detection due to many missing neighboring measurements caused by the window shield, consequently, the green measurement will be assigned a different cluster.

Other causes are partial occlusion when objects block others behind them, or when the laser hits a surface at a very steep angle and reflects away. Additionally, if an object is too far away, beyond the sensor's maximum range, measurements become harder to capture [1]. The non-uniform vertical resolution in the LiDAR sensor can magnify the effect of the previously mentioned problems. This is because some parts of the environment, especially in regions between the non-uniform sparse beams, are sampled more sparsely or even missed entirely. As a result, certain surfaces may fall between laser layers and not be hit by any beam at all, leading to gaps in the range image.

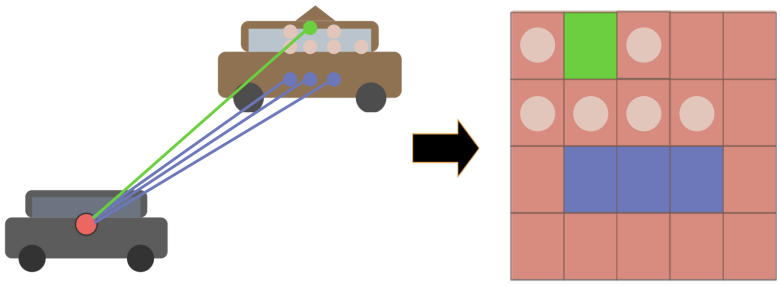


Figure 4.5: Example of missing LiDAR measurements caused by a transparent surface, like a car window. The three blue beams (left image) return valid points, shown in blue in the range image. The green beam also returns a valid point, but due to missing neighboring points around it, it gets separated from the blue points during clustering. Red squares correspond to other cells in the range image coming from other beams not highlighted in the scenario to the left.

A different situation is highlighted in Figure 4.6, where a LiDAR beam (purple) may strike an object at a relatively tight angle, but the subsequent beams (yellow) hit distant objects. As a result, these points are mapped neighbors in the range image even though they are far away from each other in the 3D real world. This will lead to outliers, where points are clustered differently than other points from the same object.

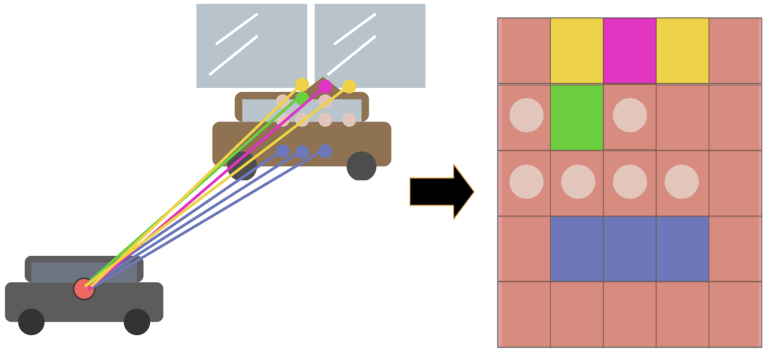


Figure 4.6: Example built on 4.5. The purple beam from the left figure hits the top part of the car, while the subsequent beams, which are highlighted, hit the wall, which is way farther away from the car. In the range image, this causes the purple point to appear next to points from a distant object, leading to incorrect clustering, even though it belongs to the same object as other car points.

4.3 Clustering Algorithms

In this section, the clustering methods developed and implemented as part of this project are discussed. These methods are an attempt to solve the problems outlined in the previous section. Two custom clustering algorithms were designed, both of which are built upon principles introduced in the work [10] (see Section 3.4.10). As a result, both strategies are based on connected components and have four neighbors; the way connectivity is calculated in the connected component stage is where they diverge.

The two clustering approaches used in this work are outlined below:

- Approach 1 – Hybrid Breakpoint and Angular Clustering (HBAC): This method combines the DBD angle criteria and ABD. In the horizontal direction, the distance threshold is dynamically adjusted using the principle of ABD, combined with an angular difference criterion based on DBD angular principles. In the vertical direction, the ABD is used with some modifications on the incidence angle to account for the non-uniformity.

- Approach 2 – Range-Guided Adaptive Clustering (RGAC): Horizontally, this approach also uses ABD for adaptive distance thresholding, but uses a fixed angular threshold instead of the angular principle of the DBD. Vertically, instead of using angular or Euclidean distance thresholding, the clustering is applied based on the range values in the vertical neighbors in the range image.

The next subsection starts by outlining the key differences between our approach and the ABD, DBD, and the baseline algorithm from [10]. Afterwards, the motivations behind our modifications in the new code implementation are presented.

4.3.1 Approach 1 – Hybrid Breakpoint and Angular Clustering

- **Difference from the Baseline Implementation:** The baseline method uses a fixed uniform angular threshold (β) for clustering. Our approach introduces a hybrid criterion. For horizontal connectivity, both ABD and DBD checks are applied, thus incorporating distance and angle. For vertical connectivity, it only relies on an adaptive distance threshold, thus, only ABD is applied.

- **Non-uniform Modified ABD Clustering:** The method adapts the vertical incidence angle threshold (λ_v) to account for the non-uniformity instead of having a fixed incidence angle. The entire non-uniform scan is divided into multiple regions, where each region contains scan lines with approximately the same elevation step in degrees. Thus, the entire non-uniform scan gets split into several approximately uniform scan regions in the elevation. The threshold λ_v is first tuned empirically in the densest region, then scaled for other regions as:

$$\lambda_v^{\text{adjusted}} = \lambda_v \times \frac{\Delta\theta_{\text{dense}}}{\Delta\theta_{\text{region}}}, \tag{4.4}$$

where $\Delta\theta_{\text{region}}$ is the elevation step of the current region, and $\Delta\theta_{\text{dense}}$ is the elevation step of the densest region. This scaling ensures that the maximum allowed distance ($d_{\max}$) is adaptive to both range and elevation resolution. This results in a more consistent clustering thresholding performance across all elevations.

This adjustment is precomputed (before clustering) and stored in a lookup table. During clustering, each point's elevation region is determined, and the corresponding $\lambda_v^{\text{adjusted}}$ is fetched from the lookup table and is applied.

- **Key Difference Compared to DBD:** In the azimuth direction, points are primarily clustered based on their distance, following the ABD method. If the distance criteria fail, an additional check based on the angle formed by the three nearest neighbors is applied. However, unlike the approach in [5], clustering is not separated into distance-based and angle-based stages; both checks are integrated into a single decision step.

The code for approach 1 (HBAC) can be seen in Algorithms 3, 4 and 5.

Algorithm 3: AdaptiveThreshold(r, α, λ, σ)

$d_{max} \leftarrow \frac{r \cdot \sin(\alpha)}{\sin(\lambda - \alpha)} + 3\sigma$

return d_{max}

Algorithm 4: Hybrid Breakpoint and Angular Clustering (HBAC)

Range image R with dimensions $M \times N$

Parameters:

λ_h: Horizontal incidence angle threshold

λ_v: Vertical incidence angle threshold

σ: Sensor noise standard deviation

θ_o: Angular difference threshold for DBD (

Output: Label matrix L where $L(r, c)$ = cluster ID for point $R(r, c)$

Begin:

Initialize $L \leftarrow 0$, `current_label` $\leftarrow 1$

Convert thresholds to radians:

$\lambda_h \leftarrow \lambda_h \cdot \pi/180$, $\lambda_v \leftarrow \lambda_v \cdot \pi/180$, $\theta_o \leftarrow \theta_o \cdot \pi/180$

foreach *point* $R(r, c)$ *in the range image* **do**

  if $R(r, c)$ *is valid and* $L(r, c) = 0$ **then**

    Perform HBAC clustering from (r, c)

    Increment `current_label` after clustering completes

return L

Algorithm 5: HBAC Clustering Function from seed point (r_0, c_0)

Initialize queue Q with starting point (r_0, c_0)
$L(r_0, c_0) \leftarrow$ `current_label` `// label seed point with current label`
while *Q is not empty* **do**
  Current point $(r, c) \leftarrow Q$.pop()
  foreach *neighbor $(\Delta r, \Delta c)$ in 4-connected neighborhood* **do**
    $(r_n, c_n) \leftarrow (r + \Delta r, c + \Delta c)$
    if *(r_n, c_n) is out of bounds or already labeled $L(r_n, c_n) \neq 0$* **then**
      continue
    if *$R(r_n, c_n)$ is empty or $|R(r_n, c_n).r - R(r, c)r| > threshold$* **then**
      Search up to max_gap steps in direction $(\Delta r, \Delta c)$
      if *no valid point found or $|R(r_n, c_n).r - R(r, c).r| > threshold$* **then**
        continue
      else update (r_n, c_n) `// first valid point found`
    $p \leftarrow R(r, c)$, $q \leftarrow R(r_n, c_n)$ `// Current point and neighbor`
    if $\Delta c \neq 0$ `// horizontal neighbor` **then**
      Compute Euclidean distance $\delta \leftarrow \|q - p\|$
      $\alpha_h \leftarrow$ sensors azimuth angular resolution
      $d_{max} \leftarrow$ `AdaptiveThreshold`$(p.r, \alpha_h, \lambda_h, \sigma)$
      if $\delta < d_{max}$ `// ABD condition met` **then**
        $L(r_n, c_n) \leftarrow$ `current_label`, Q.push(r_n, c_n)
        continue
      .
      `// Apply DBD angle check since ABD failed`
      Else `// Identify horizontal neighbors from the scan line`
      $c_{prev} \leftarrow c - 1$, $c_{next} \leftarrow c + 1$
      if $0 \leq c_{prev} < N$ *and $R(r, c_{prev})$ is valid* **then**
        $p_{prev} \leftarrow R(r, c_{prev})$
      if $0 \leq c_{next} < N$ *and $R(r, c_{next})$ is valid* **then**
        $p_{next} \leftarrow R(r, c_{next})$
      if *both exist* **then**
        $\theta_1 = angle(p_{\text{prev}} \rightarrow p)$, $\theta_2 = angle(p \rightarrow p_{\text{next}})$
        if $|\theta_1 - \theta_2| < \theta_o$ # DBD condition met **then**
          - Label p_{next}: $L(r_{next}, c_{next}) \leftarrow$ `current_label`
          Enqueue p_{next}: Q.push(r_{next}, c_{next})
          Label p_{next}: $L(r_{prev}, c_{prev}) \leftarrow$ `current_label`
          Enqueue p_{prev}: Q.push(r_{prev}, c_{prev})
    if $\Delta r \neq 0$ `// vertical neighbor` **then**
      $\delta \leftarrow \|q - p\|$
      $\alpha_v \leftarrow |p.\phi - q.\phi|$
      $d_{max} \leftarrow$ `AdaptiveThreshold`$(p.r, \alpha_v, \lambda_v, \sigma)$
      if $\delta < d_{max}$ **then**
        $L(r_n, c_n) \leftarrow$ `current_label`, Q.push(r_n, c_n)

The code starts off in Algorithm 4 with the initialization of a label matrix L. The role of this matrix is to keep track of cluster labels for each cell or LiDAR point. Furthermore, the clustering process iterates over each valid non-empty point in the range image. If a point has not yet been assigned to a cluster, it becomes the seed for a new cluster. Afterwards, the breadth-first search (BFS) expands from the seed point and labels connected points that fulfill the clustering criteria in Algorithm 5.

Algorithm 5 starts off by implementing neighbor exploration and some form of gap handling on lines 5-13. The algorithm examines the four immediate neighbors above, below, left, and right of each seed point. However, the process doesn't stop if one of these nearby positions has invalid data (a gap, as covered in Section 4.2.2) or if the range difference between the neighboring points is larger than a threshold, which suggests there is partial occlusion. Instead, it keeps searching in the same direction for a predetermined maximum number of steps or until a valid point has been found. This method enables neighboring points that are separated by missing measurements to still be considered for clustering criteria. Thus, decreasing the impact of the missing points, over-segmentation problems, and also preserving the neighborhood relationship of these cells.

Moreover, the horizontal cluster is performed on lines 15 to 37. First, the algorithm checks neighboring points along the horizontal direction (lines 15–18). Thereafter, on lines 19-21 an ABD thresholding criterion is applied. This threshold is calculated by the AdaptiveThreshold function (Algorithm 3) using the point's depth, sensor resolution, and noise level. However, if the ABD condition is not fulfilled, the three neighboring points are being reconsidered based on the angle criteria of the DBD, as can be seen on lines 23-27.

This criterion is more flexible than just relying on β as in the baseline approach. If the angle difference is small (lines 31-35), the points are still clustered into the same cluster even though the ABD criteria failed. This is expected to improve performance in the scenarios described in Section 4.2.1.

For vertical clustering on lines 36-41, ABD is applied but with parameters tuned for vertical resolution and non-uniformity as explained in equation (4.4). Nevertheless, the vertical angular resolution α_v that ABD requires is calculated on line 38. This resolution is set to be the difference between the elevation of the current point and the vertical neighbor (up or down).

Furthermore, both the ABD horizontal and vertical clustering account for sensor noise (represented as Gaussian with standard deviation σ) as highlighted in equation 3.3. The output of HBAC algorithm is a labeled matrix with a cluster ID for every cell.

4.3.2 Approach 2 – Range-Guided Adaptive Clustering

The RGAC algorithm is designed to improve efficiency by reducing the trigonometric computations as well as eliminating the need for the precomputation step of the vertical elevation regions required in HBAC.

Algorithm 6: Range-Guided Adaptive Clustering (RGAC)

Range image R with dimensions $M \times N$ **Parameters:**
  λ_h: Horizontal incidence angle threshold (degrees)
  λ_v: Vertical incidence angle threshold (degrees)
  σ: Sensor noise standard deviation
  θ_h: Horizontal angular difference threshold (degrees)
  τ_v : Vertical depth threshold
Output: Label matrix L where $L(r, c)$ = cluster ID for point $R(r, c)$
Begin:
Initialize $L \leftarrow 0$, `current_label` $\leftarrow 1$
Convert thresholds to radians:
$\lambda_h \leftarrow \lambda_h \cdot \pi/180$, $\lambda_v \leftarrow \lambda_v \cdot \pi/180$, $\theta_h \leftarrow \theta_h \cdot \pi/180$
foreach *point* $R(r, c)$ *in the range image* **do**
  if $R(r, c)$ *is valid and* $L(r, c) = 0$ **then**
    Perform RGAC clustering from (r, c)
    Increment `current_label` after clustering completes
return L

Algorithm 7: RGAC Clustering function from seed point (r_0, c_0)

```
1  Initialize queue Q with starting point (r_0, c_0)
2  L(r_0, c_0) ← current_label
3  p ← R(r_0, c_0)
4  Vertical DEPTH threshold τ_v
5  while Q is not empty do
6      Current point (r, c) ← Q.pop()
7      p ← R(r, c)
8      foreach neighbor (Δr, Δc) in 4-connected neighborhood do
9          (r_n, c_n) ← (r + Δr, c + Δc)
10         if (r_n, c_n) is out of bounds or already labeled L(r_n, c_n) ≠ 0 then
11             continue
12         if R(r_n, c_n) is empty or |R(r_n, c_n).r − R(r, c)r| > threshold  // Handle
            gaps (missing data points in the range image) then
13             Search up to max_gap steps in direction (Δr, Δc)
14             if no valid point found then
15                 continue
16             else update (r_n, c_n)                    // first valid point found
17         q ← R(r_n, c_n)
18         d_1 ← max(p.r, q.r), d_2 ← min(p.r, q.r)
19         if Δc ≠ 0                                  // horizontal neighbor then
20             α_h ← sensors azimuth angular resolution
21             δ ← ‖q − p‖                                  // Euclidean distance
22             τ_h ← AdaptiveThreshold(p.r, α_h, λ_h, σ)
23             Compute angle β ← arctan((d_2 · sin α_h) / (d_1 − d_2 · cos α_h))
24             if β > θ_h and 0 < δ < τ_h     // Range-guided angular test then
25                 L(r_n, c_n) ← current_label, Q.push(r_n, c_n)
26         if Δr ≠ 0                                    // vertical neighbor then
27             δ ← |d_1 − d_2|
28             if δ < τ_v                    // Range-guided distance test then
29                 L(r_n, c_n) ← current_label, Q.push(r_n, c_n)
```

The RGAC Algorithm 6 also starts with initialization and threshold conversion, followed by a sequential reading of the cells of the range image. Clustering is occurring in the RGAC clustering function (Algorithm 7) handles neighbor exploration and gap handling like HBAC in lines 5-16.

Horizontal clustering is being done on lines 19-25. Unlike HBAC, which considers the angles between three points, in RGAC it is simplified by using a fixed angle threshold θ and retaining the ABD (distance) criterion. However, in RGAC, both conditions must be satisfied to form a cluster, while in HBAC, only one condition (either distance or angle) must be fulfilled. Additionally, only one horizontal neighbor is evaluated instead of two, which further reduces the computational cost.

A vital modification to vertical thresholding is the switch from ABD to a straightforward distance range threshold, which shortens computation times and makes the system less prone to the non-uniformity problem. The reason is that ABD requires some trigonometric operations, which usually require more computation time. Additionally, since the elevation is not uniform, it might result in instability and some misclustering in the results because the α_v in the HBAC is constantly changing. This is analogous to saying in ABD terminology that the maximum allowed circular region or d_{max} is inconsistent for points at the same range.

The reason behind range-based thresholding is that it does not require extra computation, nor is it affected by the non-uniformity. As illustrated in Figure 4.7, vertically adjacent points on a continuous surface generally have similar range values, making range a reliable basis for clustering in the vertical direction. Although rare, if multiple objects are stacked on top of one another while having a similar range, there will still be gaps and shifts in the range image. Thus, they will still be clustered differently because of the gap between them. Unlike Euclidean distance, range differences between points on the same surface remain consistent with depth. Consequently, this fixed threshold is not dependent on the changes in the vertical step size (i.e., distance between points), thus the non-uniformity problems in the vertical angular resolution are mitigated.

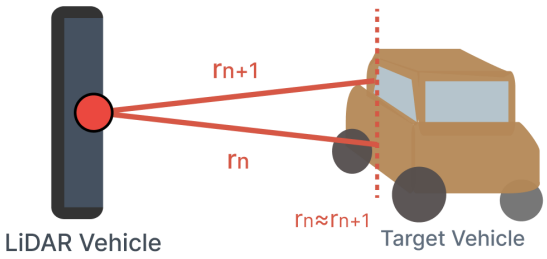


Figure 4.7: Illustration showing that vertically neighboring points have similar range values, indicating they lie on the same surface.

4.3.3 Merging process with KD Trees

To address the problem of the outliers (i.e., isolated points caused by missing measurements), K-D trees are used to refine and merge these points as a post-processing step.

The method involves collecting all clusters that are smaller than a predetermined maximum size threshold and marking them as potential merging candidates. For each candidate, the surrounding points within a circular 3D region are examined. The candidate is merged to the dominant cluster if more than τ% of the points in the circular region belong to that dominant cluster, where τ is a predefined threshold. Thus, the merging decision is transferred from 2D range image-based neighborhood criteria to the 3D spatial proximity. This mitigates the problems introduced by the range image representation.

4.3.4 HDBSCAN Baseline Comparison

HDBSCAN is used as a comparison method for evaluating the proposed clustering approach. HDBSCAN is a density-based clustering method rather than a range-image-based clustering, providing valuable insights into an alternative approach for 3D point cloud segmentation.

Unlike DBSCAN, which uses a fixed density threshold, HDBSCAN extends density-based clustering by performing hierarchical analysis across varying density levels to automatically determine optimal cluster boundaries (i.e., automatically tune its parameters). The algorithm transforms the input space using mutual reachability distances, constructs a minimum spanning tree, and builds a cluster hierarchy. Stable clusters are then extracted based on persistence measures across different density thresholds. This hierarchical approach enables HDBSCAN to handle clusters of varying densities and automatically determine the number of clusters without prior specification.

The HDBSCAN algorithm is implemented using the standard scikit-learn library with default parameters, applied to the same preprocessed 3D point cloud data for fair comparison with the proposed method.

5

Evaluation Strategy

In this section, two types of evaluations, a quantitative evaluation and a visual examination, are presented as well as the design decision of the evaluation methods will be reasoned. The rationale behind the selection of particular situations in the visual evaluation, as well as the selection process and limitations of the quantitative data set will be provided. Furthermore, the evaluation metrics for the quantitative tests including their disadvantages and advantages are presented.

5.1 Visual Evaluation

The goal of the visual evaluation is to test HBAC and RGAC on challenging edge cases where other scenarios might struggle. Unlike the quantitative evaluation, this approach allows full control of exactly how complex the scenarios should be, such as dense urban scenes with tight object angles (i.e., the angle between two points on respective objects is small, discussed in Section 4.2.1) and varying weather conditions (e.g., rain and snow). In contrast, the quantitative hand-annotated data is only available for dynamic objects (e.g., cars or trucks) under favorable weather conditions.

Six representative scenarios are selected as highlighted in Figure 5.1. Scenario 5.1a shows three cars close together, two of which were positioned with have small angular separation. In addition, there are two cars parked extremely near to each other on the left and several distant cars. Scenario 5.1b has many closely positioned and parallel cars behind each other on both sides of the street. This is testing the problem highlighted in Section 4.2.1.

To evaluate the flexibility of both the angle and the distance thresholding, a rainy scenario 5.1c is selected, which has a high density of pedestrians near poles. Scenario 5.1e presents a cluttered environment with various vehicle such as cars, trucks and vans positioned closely together near stationary landmarks (e.g., Fuel Station's poles and different parts), forming large connected objects. Finally, scenarios 5.1d and 5.1f include bad weather conditions, which introduces additional sensor noise that may degrade the accuracy of the clustering methods.

(a) Three nearby cars with tight angles and dense layout

(b) Parallel cars on both sides of a narrow street

(c) Rainy scene with pedestrians and poles at tight angles

(d) Heavy rain with challenging parallel vehicle layout

(e) Dense traffic: cars, truck, and van packed closely

(f) Rainy scene with noise and parallel vehicle layout

Figure 5.1: Six complex evaluation scenarios used for testing clustering robustness.

5.2 Quantitative Evaluation

The quantitative evaluation metrics are computed by matching algorithm clustering results to ground truth segments and measuring their overlap. A description is provided below of how the scores are calculated, including how overlapping ground truth annotation are handled.

5.2.1 Over/Under-Segmentation Metrics

In the definition of Section 3.3.1, over-segmentation as several detections covering a single ground truth object, while under-segmentation is when a single detection covers multiple ground truth objects. The evaluation method is inspired by the approach described in [27].

The idea is to find the set of points A_{gt} inside a given ground truth bounding box. For each cluster s, let A_s be the set of points belonging to that cluster. The next step is to find the predicted cluster set A_s that maximises

$$\arg\max_{s'} |A_{s'} \cap A_{\text{gt}}| \,. \tag{5.1}$$

Thereafter, the over/under-segmentation score will be computed using the best matched cluster set A_s. The under-segmentation score is evaluated using the ratio of the intersection set ($|A_s \cap A_{gt}|$) to the set of the best matched cluster ($|A_s|$). The equation for under-segmentation is x.

$$U = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{|A_s^{(i)} \cap A_{gt}^{(i)}|}{|A_s^{(i)}|} \right), \tag{5.2}$$

where N is the total number of ground truth bounding boxes, $A_{gt}^{(i)}$ is the i-th ground truth object, and $A_s^{(i)}$ is the corresponding segmented cluster.

Alternatively, the over-segmentation score is obtained by comparing the intersection set $|A_s \cap A_{\text{gt}}|$ to the set of the ground truth bounding box points $|A_{\text{gt}}|$. The equation for over-segmentation is

$$O = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{|A_s^{(i)} \cap A_{gt}^{(i)}|}{|A_{gt}^{(i)}|} \right). \tag{5.3}$$

It is important to note that if two ground truth bounding box sets of points overlap (i.e., the same points are in two different bounding box sets at the same time), over- and under-segmentation scores are not computed, as it becomes ambiguous which bounding box the shared points belong to. In such cases, both bounding boxes are excluded from evaluation. Although rare, this situation may occur when, for example, two pedestrians are close to one another. Nevertheless, these scores have values between zero and one, where one indicates better clustering performance, reducing over- and under-segmentation errors.

5.2.2 Data Selection

A comprehensive internal dataset from Zenseact which is consists of LiDAR point clouds accompanied by 3D bounding box (in spherical coordinates) annotations is utilized in the experimental evaluation.

A systematic processing methodology to transform this raw data into a format suitable for evaluation (e.g., coordinate transformation and the conversion of bounding box annotations into point-wise labels) was developed.

In Figure 5.2 the spatial distribution of objects and the correspondence between the clusters generated by the algorithm and the ground truth annotations are shown.

Figure 5.2: Visualization of Point Cloud with Ground Truth Annotations. Colored points represent different clusters. Semi-transparent overlays highlight ground truth object labels derived from bounding boxes.

5.2.3 Quantitative Evaluation Limitations

Despite systematic preparation for dataset, several inherent limits influence quantitative measurement:

Bounding Box Approximation. For complex objects in real world, bounding box-based ground truth annotations simplify through geometric approximation. The Intersection of Union (IoU) metric is effective when objects are well represented by rectangular bounding boxes, but it may cause false results for complex-shaped objects [44], such as pedestrians which have irregular shapes, leading to evaluation biases.

Limited Field of View. The annotation may not match the observed results of LiDAR due to some areas are not visible to the sensor. For example only the rear half of a car is scanned, so that a small bounding box is generated based on it. While the real ground truth annotation takes the whole length of the vehicle, including the invisible front half, which makes traditional metrics like IoU not fit for evaluation in some cases.

Point Density Variation. Objects at different distances and vertical positions relative to the sensor exhibit variations in point density, as shown in Figure 5.2. In addition, due to sensor noise, edge points often have low accuracy. This non-uniform density influnces clustering performance and complicates the evaluation of the entire scene.

Class Imbalance. The dataset has substantial number of certain object classes like vehicles, which raises the evaluation results with the dominant classes. Although this imbalance reflects the real scenarios, it also complicates the algorithm analysis

5.2.4 Metrics Selection

Taking these limitations into consideration, an evaluation framework is developed that focus on point-level analysis over traditional bounding box metrics.

Using bounding box IoU, which has the problems described above, instead point-level metrics as defined in Equations (5.2) and (5.3) are adopted. Compared to the problems within the IoU metric for bounding boxes:

- It assumes objects are represented by their bounding box geometry well.
- It is sensitive to small orientation differences that may be meaningless.
- It fails to show internal structure and point density variations.

The point-level metrics directly measure how well the algorithm assigns points to clusters that correspond with ground truth annotation. With the more detailed evaluation addresses the limitations in Section 5.2.3.

For each ground truth segment, the best matching clustering result is identified based on maximum point overlap, as described in Equation (5.1). The approach provides a fair comparison even when the clustering algorithm produces multiple clusters for a single ground truth object or when a single algorithm cluster covers multiple ground truth objects. It shall be noted that the spatial scope of clustering might deviate from the ground truth with no significant influence on the evaluation accuracy.

In addition, the evaluation framework preserves class information from ground truth annotations, helping to analyze clustering performance for specific classes. The approach helps in evaluating the performance of the algorithm on various object types separately.

5.2.5 Alternative Approach

Within the process of evaluation framework development, alternative approaches are considered but impractical for the use case.

An optimal approach is point-level manual annotation of the entire dataset. It proved impractical nevertheless, due to several reasons: labelling millions of points across hundreds of frames takes a great amount of time; inconsistencies between multiple annotators; and even human experts sometimes struggle with defining partial occlusion and imprecise object borders. Hand annotation proved to be ineffective compared to automated approaches.

The dataset selection and assessment methodology with point-level metrics providing a better insights into algorithm performance than traditional bounding box, particularly given the complex objects and varying conditions present in our dataset.

6

Results

The findings from the Evaluation Strategy Chapter 5 will be presented in this chapter. Each case will be shown, and the most significant findings will be highlighted. Since ground removal and the K-D tree merging step are pre- and post-processing procedures used for all baseline, HBAC, and RGAC techniques, the chapter will start by highlighting their outcomes. The findings of the visual evaluation will then be displayed, followed by the results of the quantitative evaluation. Finally, a computation time evaluation will be conducted.

6.1 Ground Removal Results

To efficiently cluster point clouds, ground removal is an important preprocessing step, as ground points will connect spatially separated objects, resulting in over-segmentationissue. Figure 6.1 demonstrates the performance comparison between our fixed threshold and adaptive ground removal approaches.

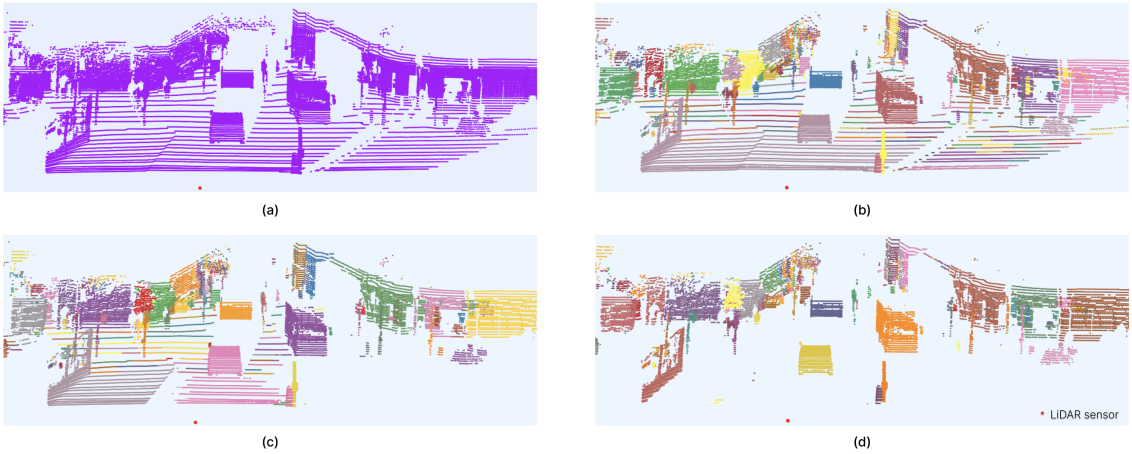


Figure 6.1: Ground removal comparison. (a) Original point cloud with ground points (purple). (b) Clustering result without ground removal. (c) Fixed threshold ground removal. (d) Dual-grid adaptive ground removal.

Figure 6.1a depicts the original LiDAR point cloud, highlighted in colour purple. The ground occupies a part of the scene, including the up-and-down main roads and sidewalks. When applying clustering while retaining the ground, as shown in Figure 6.1b, due to the connection of ground points, multiple targets are wrongly clustered

as a whole; each colour is designed to represent a clustering result within the point cloud scenario.

The fixed threshold method in Figure 6.1c removes ground points in right part of the areas with consistent elevation but fails in regions with varying terrain levels. The dual-grid adaptive method is shown in Figure 6.1d. It can effectively remove the ground points in the entire scene regardless of elevation variations, laying the foundation for clearer separation of dynamic objects in downstream clustering.

6.2 K-D Tree Merging Step

The following results highlight the effect of K-D trees on merging small clusters with the most frequent closest cluster. The effect is shown in Figure 6.2, where part e highlights the pre-merging step, where only a few individual points below the car and on top of the car get merged to the larger cluster, which can be seen in part d.

This merging step, as explained in Section 4.3.3, happens for clusters with fewer than three points, which is the predetermined maximum size threshold. As observed, the merging succeed without affecting the original large clusters. This postprocessing step is applied after initial clustering and does not alter the main cluster structure. All results presented in this chapter include this step (except the HDBSCAN).

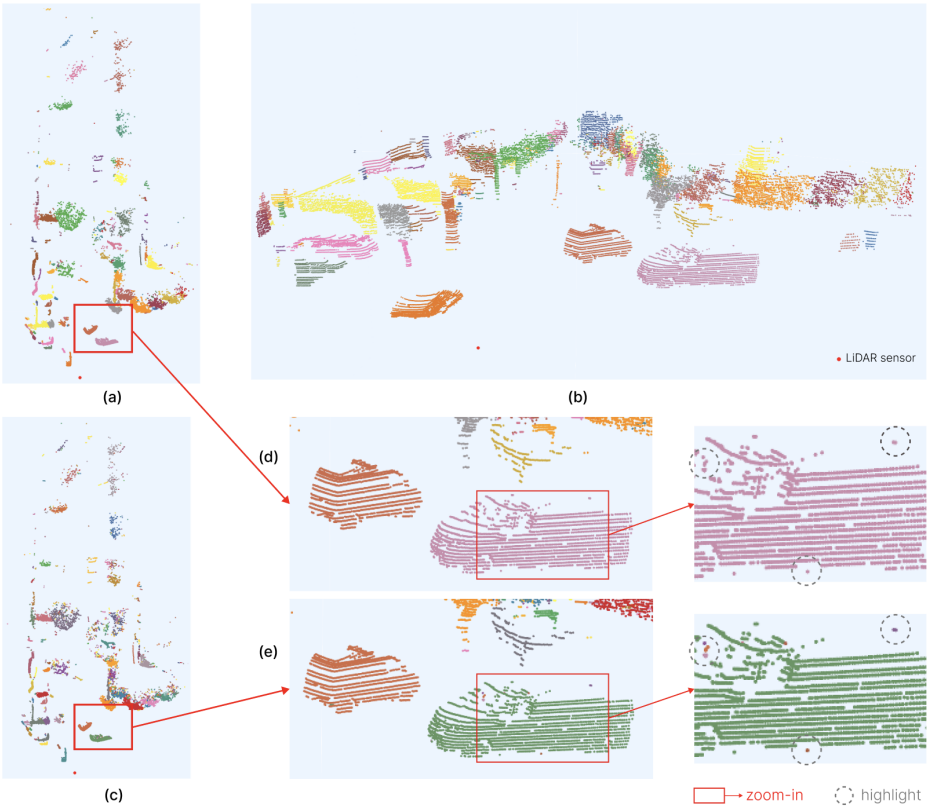
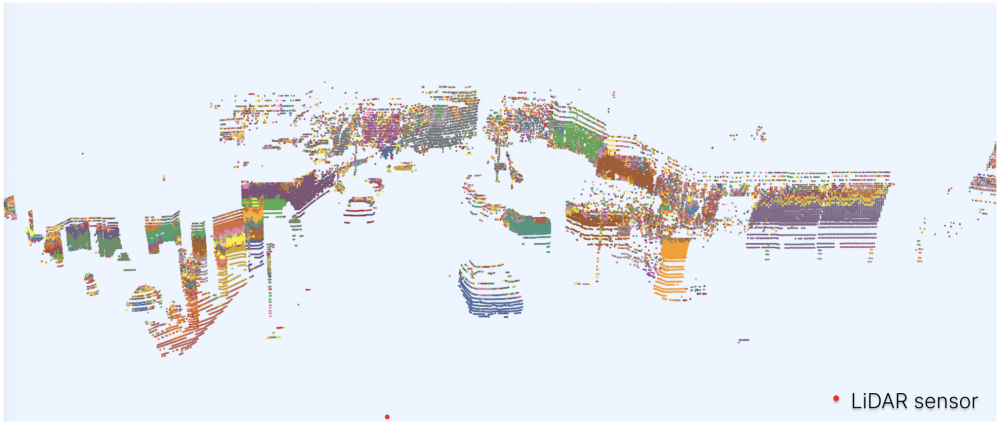


Figure 6.2: Clustering results in urban road section of Scenario 1 driving. (a),(b) Bird-eye/front view with merge; (c) Bird-eye view without merge; (d),(e) Zoomed-in key section with/without merge.

6.3 Baseline Threshold Determination

A systematic evaluation of the threshold parameters described in the baseline model is performed. The aim is to identify the optimal threshold configuration by implementing multiple sets of parameter threshold experiments.



(a) $\theta_{baseline} = 1$

(b) $\theta_{baseline} = 2$

(c) $\theta_{baseline} = 10$

Figure 6.3: Comparison of different threshold values for baseline model.

As shown in Figure 6.3, we present the performance of three representative threshold values: $\theta_{baseline} = 1$, $\theta_{baseline} = 2$ and $\theta_{baseline} = 10$ on the scenario. It is worth noting that the original baseline model uses the $\theta_{baseline} = 10$ threshold value and the results are shown in Figure 6.3c. However, through quantitative analysis and comparative experiments, the $\theta_{baseline} = 2$ threshold value achieves superior performance by providing the best balance between over-segmentation and under-segmentation compared to smaller or larger parameters. Based on the comprehensive evaluation of the experimental results, $\theta_{baseline} = 2$ threshold value was selected as the optimal configuration of the baseline model to serve as the control group, and this choice was further verified in the subsequent experiments.

6.4 Visual Evaluation

This section presents a comprehensive visual analysis of the clustering performance across six different scenarios using four different approaches: baseline implementation, HDBSCAN, RGAC, and HBAC. The evaluation scenarios are systematically organized into two categories: typical urban road sections (Scenarios 1-3) and rainy urban environments (Scenarios 4-6).

6.4.1 Urban Road Section Scenarios

Scenario 1: Standard Urban Traffic

Figures 6.4, 6.5, and 6.6 display the clustering results for the first urban scenario. Figure 6.4 depicts the baseline implementation with a front perspective on the right and a bird's-eye view on the left. The baseline approach generates noisy results, with many points improperly grouped, resulting in considerable over-segmentation. Under-segmentation is minimized with the chosen threshold $\theta_{baseline} = 2$.

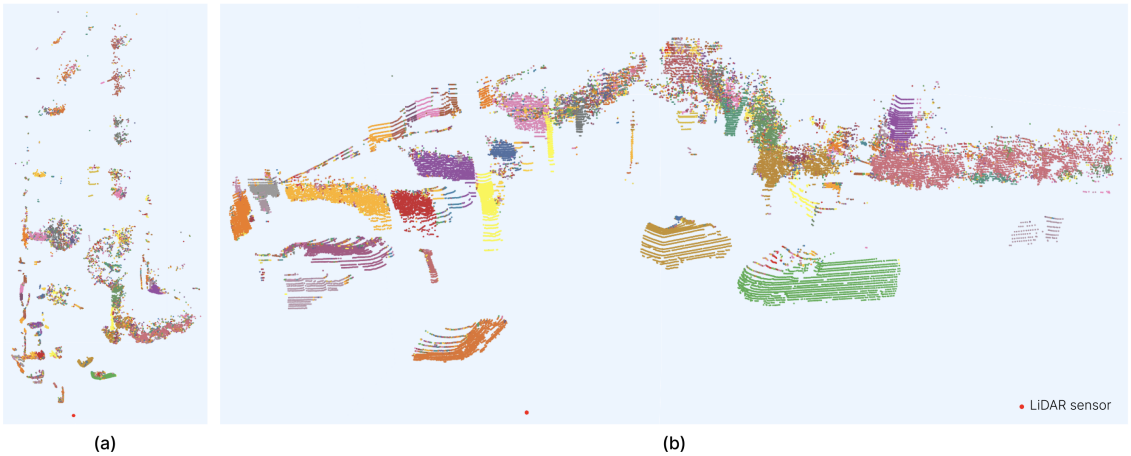


Figure 6.4: Clustering results in urban road section of Scenario 1 driving with baseline implementation. (a) Bird-eye view; (b) Front view. The color represents different clusters identified by the LiDAR sensor, and the red dot in the legend indicates the position of the LiDAR sensor.

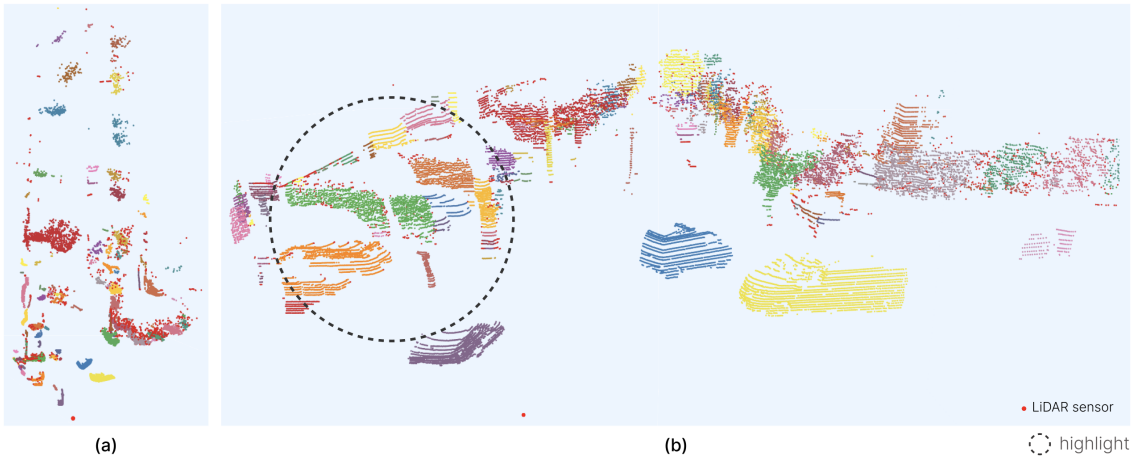


Figure 6.5: Clustering results in urban road section of Scenario 1 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

The HDBSCAN clustering results in Figure 6.5 show that larger objects are well-differentiated as separate clusters where density differences are pronounced. However, as highlighted in the circled region, HDBSCAN incorrectly merges vehicles and wall structures next to each other into the same cluster shown in orange and green, since both objects exhibit similar local point densities. At the same time, the HDBSCAN over-segments the large tree trunk in the circled region. The distant car to the right is also over-segmented into multiple clusters. However, it does not produce outliers (i.e., different clusters due to missing measurement problem) even without the merging step.

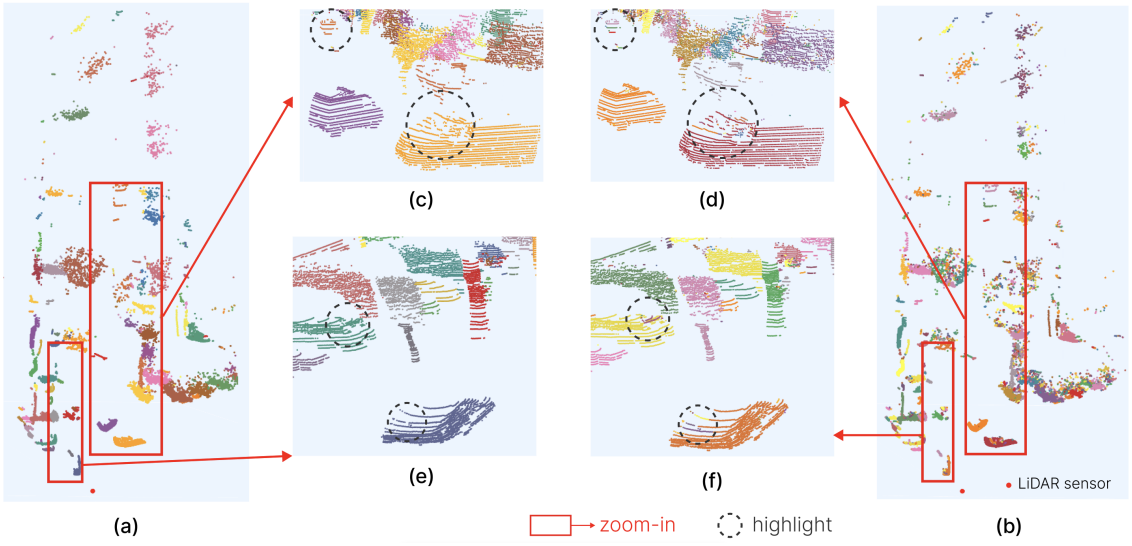


Figure 6.6: Clustering results in urban road section of Scenario 1 driving. (a),(c),(e) Results using the RGAC algorithm. (b),(d),(f) Results using the HBAC algorithm. (a),(b) Bird-eye view; (c)-(f) Zoomed-in key sections.

On the contrary, Figure 6.6 shows more promising results from the proposed algorithms. The left side (a, c & e) corresponds to the results from the RGAC approach,

while the right side (d, f & b) corresponds to HBAC. Both methods perform well in clustering, with almost everything appearing to be completely clustered. Note that the vehicles parked in the upper left region of sections e and f are well clustered, despite that they are parked at extremely tight angles to one another (i.e., the angle between two points on respective objects is small, discussed in section 4.2.1). This scenario seems to strike a great balance between over- and under-segmentation. However, the HBAC approach result introduces less stable results since there are some over-segmentation errors. An example is the circled distant car in Figure 6.6 (c & d), which appears to be clustered correctly in RGAC, but the HBAC fails to cluster.

Scenario 2: Dense Traffic Configuration

Figures 6.7, 6.8, and 6.9 represent the second urban scenario. The same observation can be derived regarding this scenario, where, for the baseline case, the results are noisy. This scenario has too many cars behind and close to each other, increasing the risk of under-segmentation.

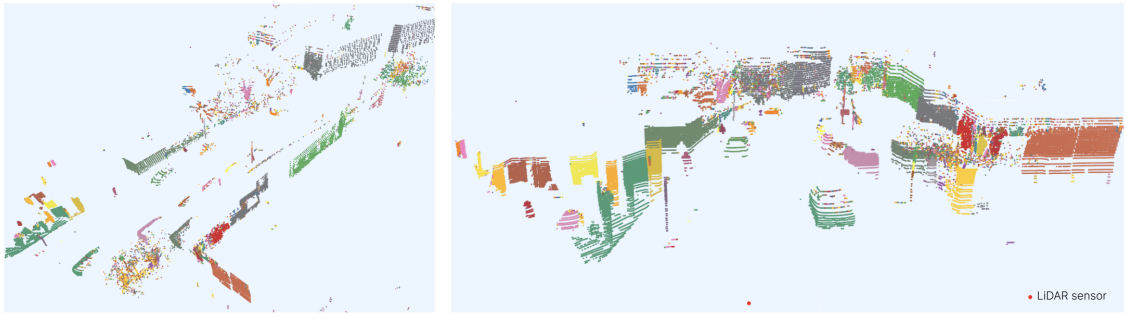


Figure 6.7: Clustering results in urban road section of Scenario 2 driving with baseline implementation. (a) Aerial oblique view; (b) Front view.

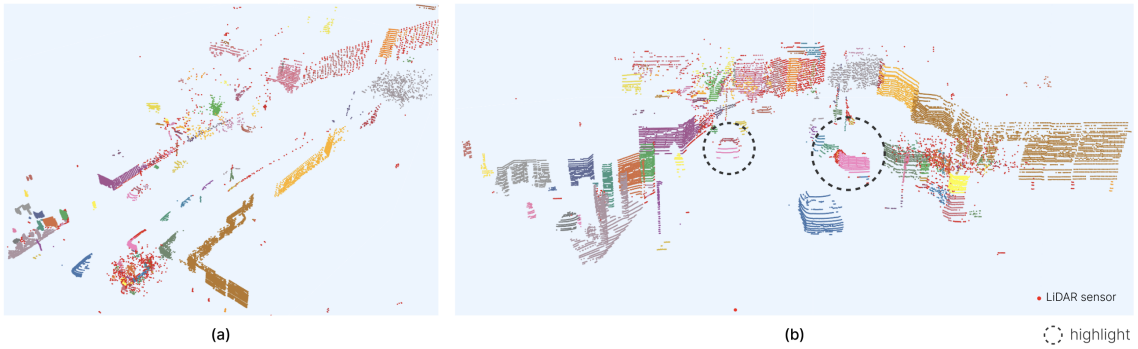


Figure 6.8: Clustering results in urban road section of Scenario 2 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

In Figures 6.8, HDBSCAN over-segments vehicles in the highlighted non-uniform scan regions. The pink van (right circle) is vertically split into two clusters at its rear section, while its hood-wheel junction is horizontally fragmented into multiple clusters. The car (left circle) is incorrectly divided into upper and lower parts separated by the windshield boundary.

For the proposed methods, as shown in Figure 6.9, the upper results (a & b) demonstrate RGAC performance, while the lower results (c & d) show HBAC performance. The RGAC approach effectively separates distantly parked adjacent vehicles, maintaining clear cluster boundaries even in dense configurations. HBAC produces comparable results but exhibits some minor over-segmentation with small extraneous clusters, particularly visible in the three highlighted circle sections. These artifacts primarily occur at the upper edges of nearby vehicles and distant vehicles, where the clustering creates small clusters along object boundaries. Overall, both methods separate each car with minimal to non-existent over-segmentation.

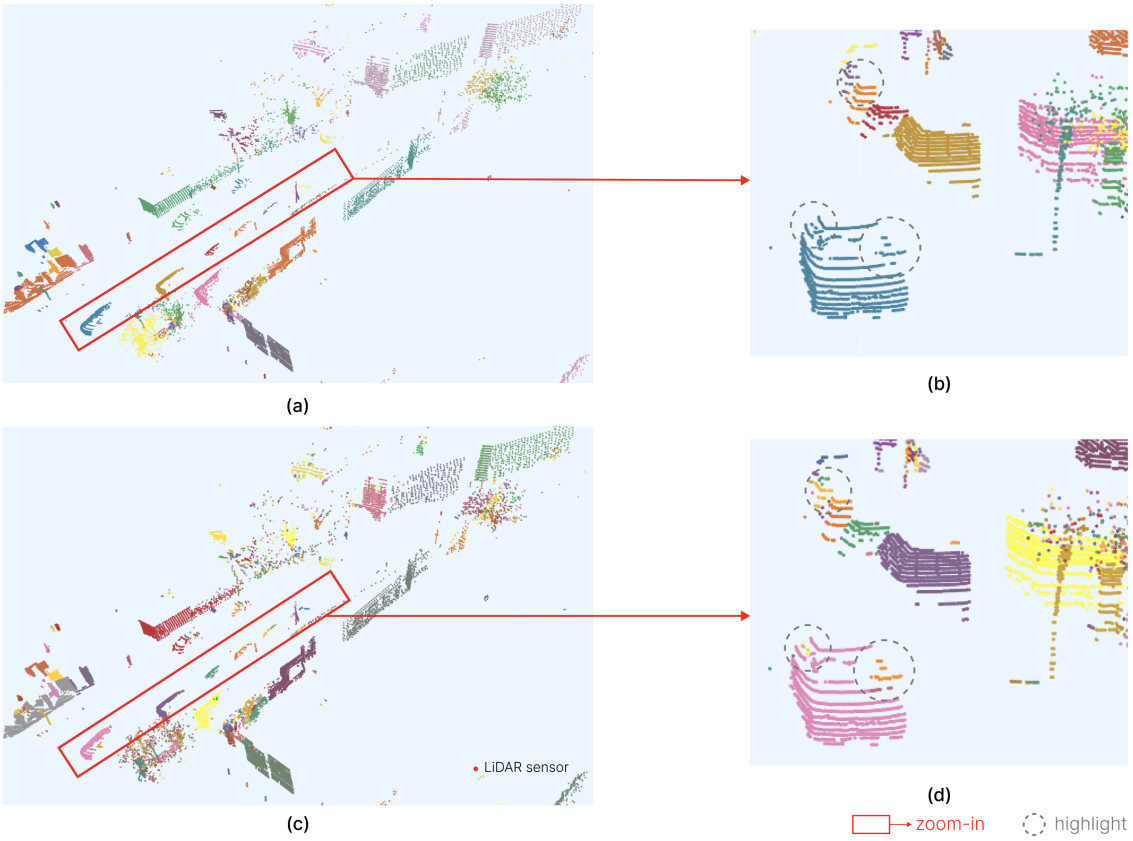


Figure 6.9: Clustering results in urban road section of Scenario 2 driving. (a),(b) Results using RGAC algorithm. (c),(d) Results using HBAC algorithm. (a),(c) Aerial oblique view; (b),(d) Zoomed-in key sections.

Scenario 3: Complex Multi-Object Environment

As for the complex case of scenario 3, the results are shown in Figures 6.10, 6.11, and 6.12. The baseline still shows random clustering, but a general pattern is noticeable where different objects are mostly assigned to separate clusters.

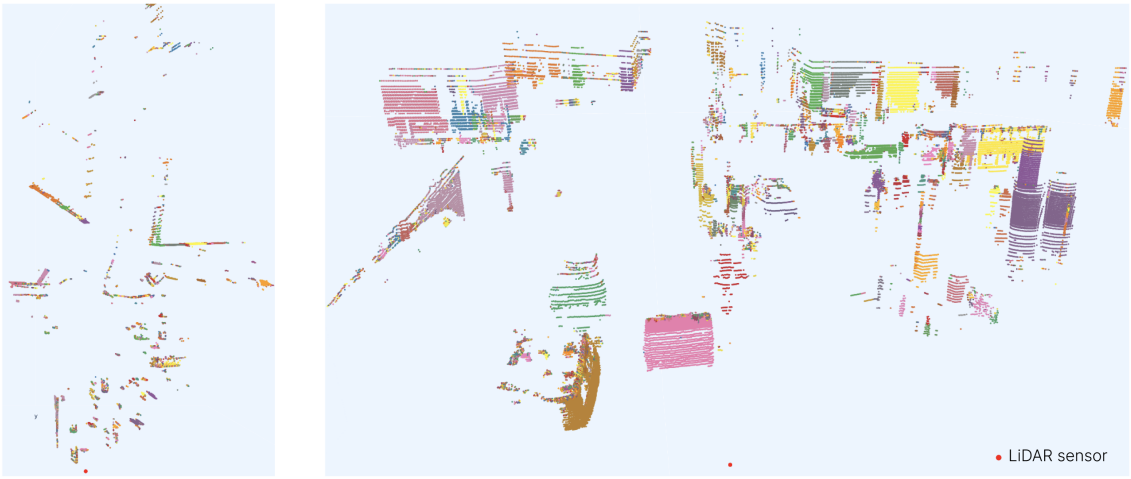


Figure 6.10: Clustering results in urban road section of Scenario 3 driving with baseline implementation. (a) Bird-eye view; (b) Front view.

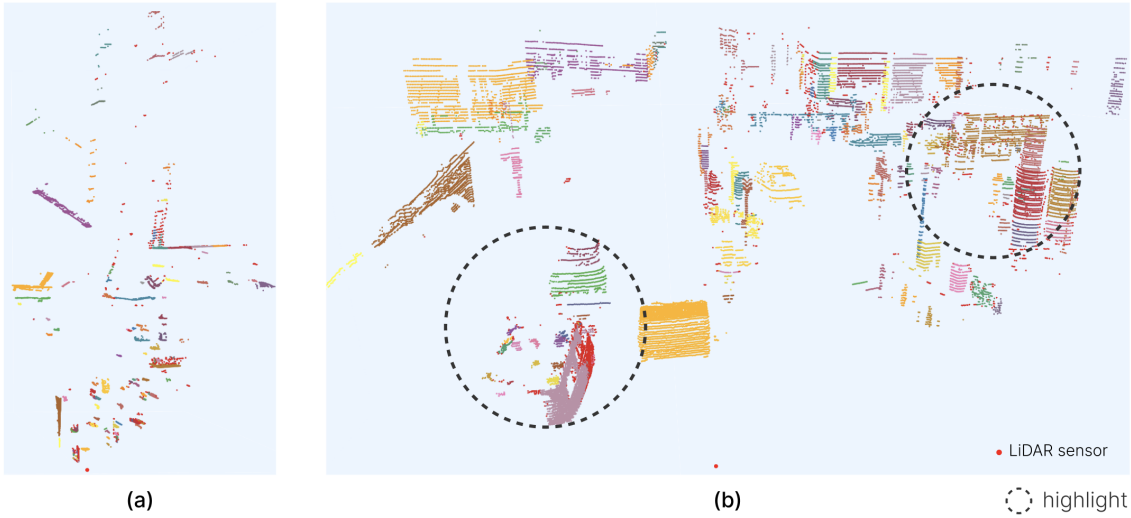


Figure 6.11: Clustering results in urban road section of Scenario 3 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

The HDBSCAN clustering results in Figure 6.11 indicate mixed performance across different regions. In the right highlighted circle, HDBSCAN successfully clusters the long double-decker bus as a single coherent unit, demonstrating good performance on large uniform objects, though numerous small clusters remain unmerged. In contrast, the building pillar in the foreground is clearly segmented into upper and lower parts, and a nearby pedestrian is similarly split horizontally at the waist. In the left highlighted circle, the closer vehicle is vertically divided into two segments,

while a more distant vehicle is over-segmented from top to bottom into three distinct parts: roof, body, and undercarriage, indicating poor clustering performance on vehicles at close and distant ranges.

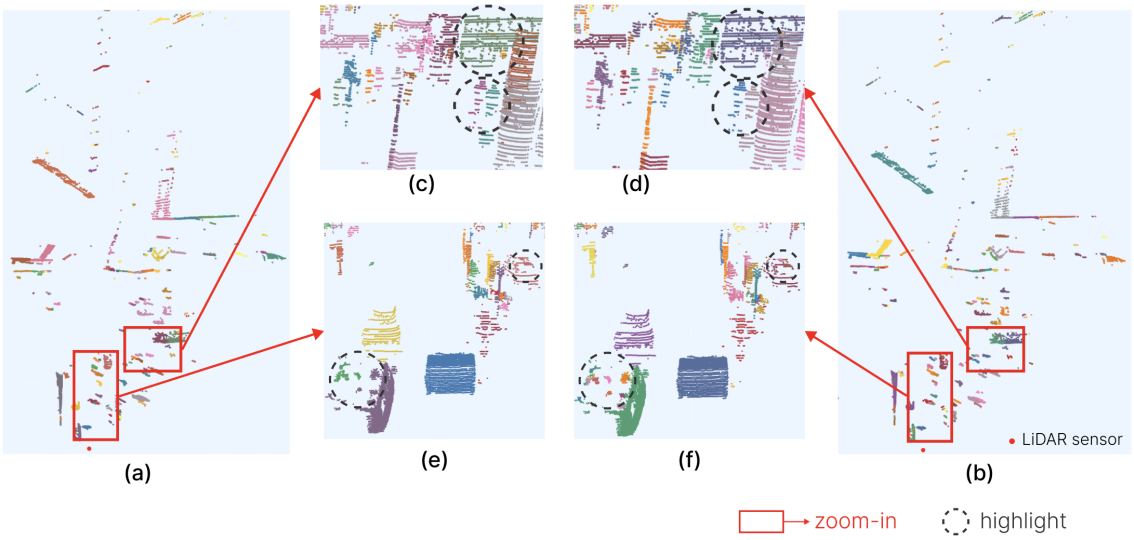


Figure 6.12: Clustering results in urban road section of Scenario 3 driving. (a),(c),(e) Results using the RGAC algorithm. (b),(d),(f) Results using the HBAC algorithm. (a),(b) Bird-eye view; (c),(f) Zoomed-in key sections.

For the results shown in Figure 6.12, it is interesting to note that both algorithms successfully manage to cluster pedestrians that are standing very close to each other in parts (c & d). However, the bus is split into two clusters due to a large gap caused by the phenomenon of partial closure from a pole blocking the LiDAR beams.

In parts (e & f), which are very dense, all poles got clustered differently from the humans. The RGAC demonstrates better clustering performance on the more distant circled vehicle than the HBAC approach. However, some noise is observed within the nearer highlighted vehicle, which is caused by LiDAR beams hitting a transparent material. In this case, it was a window, creating ghost points at incorrect spatial locations and consequently leads to poor detection quality.

6.4.2 Rainy Urban Environments Scenarios

The following three scenarios evaluate algorithm performance under adverse weather conditions, specifically rainy environments that present additional challenges for LiDAR-based clustering due to reduced visibility and atmospheric interference.

Scenario 4: Basic Rainy Conditions

The results of the rainy scenario 4 are shown in Figures 6.13, 6.14, and 6.15. This scenario tests the robustness of clustering under adverse weather conditions, where rain introduces noise and affects LiDAR data quality. The baseline method continues to exhibit poor performance with persistent over-segmentation issues across the scene as shown in Figure 6.13

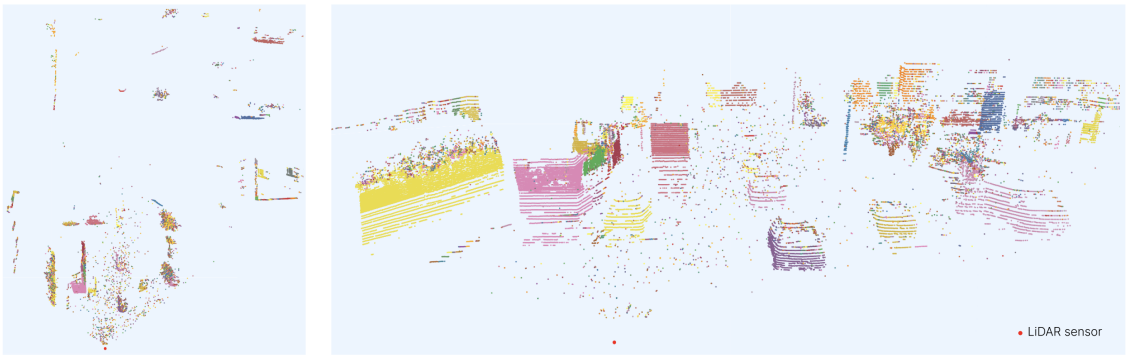


Figure 6.13: Clustering results in rainy urban road section of Scenario 4 driving with baseline implementation. (a) Bird-eye view; (b) Front view.

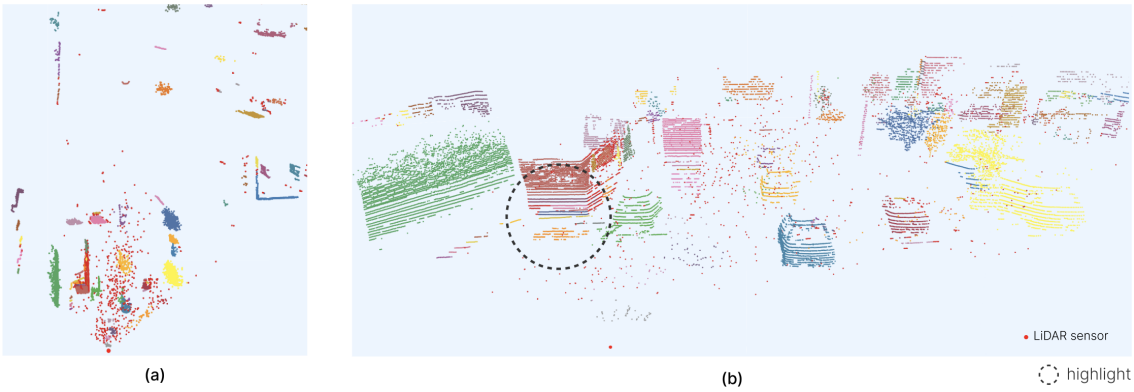


Figure 6.14: Clustering results in urban road section of Scenario 4 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

The HDBSCAN results in Figure 6.14 demonstrate generally good performance with clear separation of major objects. However, as highlighted in the left circled region, significant over-segmentation occurs due to non-uniform scan patterns. The lower half of the bus body is vertically fragmented into five distinct clusters. One section is further subdivided horizontally, illustrating how varying point densities create artificial boundaries within a single coherent object.

In contrast, both RGAC and HBAC in Figure 6.15 maintain robust clustering despite the challenging rainy conditions, demonstrating their resilience to weather-induced noise. Comparing the two proposed methods, RGAC shows consistency in maintaining object boundaries, while HBAC exhibits slightly more over-segmentation artifacts, particularly visible in the circled vehicle region in part d, where clustering creates additional small clusters along object edges.

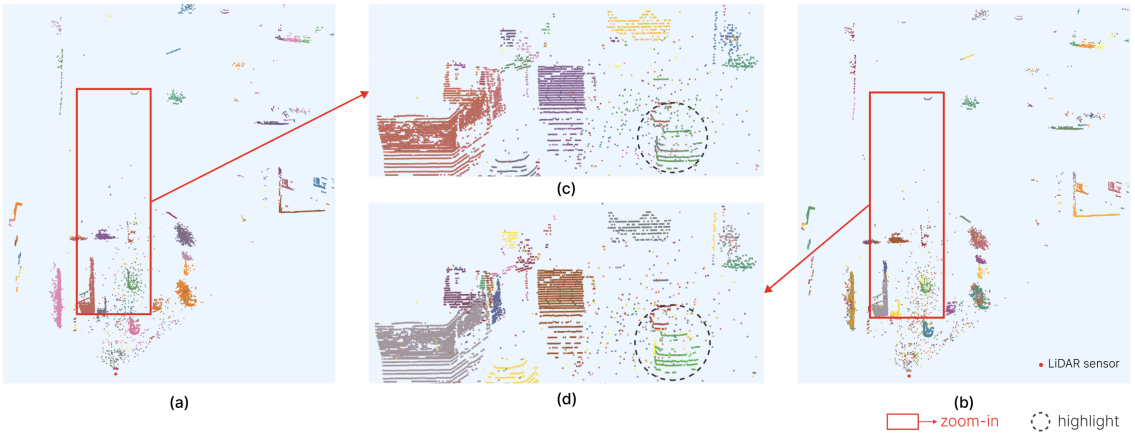


Figure 6.15: Clustering results in rainy urban road section of Scenario 4 driving. (a),(c) Results using the RGAC algorithm. (b),(d) Results using the HBAC algorithm. (a),(b) Bird-eye view; (c),(d) Zoomed-in key sections.

Scenario 5: Complex Rainy Environment

Scenario 5 presents another rainy scenario with results in Figures 6.16, 6.17, and 6.18. Notable under-segmentation occurs with pedestrians, where multiple individuals are incorrectly grouped into the same cluster despite point cloud noise.

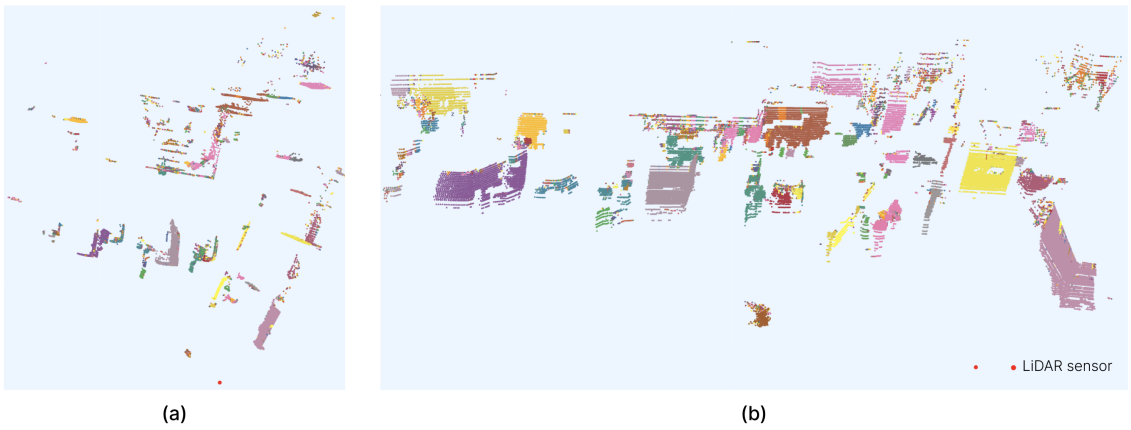


Figure 6.16: Clustering results in rainy urban road section of Scenario 5 driving with baseline implementation. (a) Bird-eye view; (b) Front view.

The HDBSCAN clustering results in Figure 6.17 exhibit significant over-segmentation issues in both highlighted circular regions. In the right circled region, the pedestrian with an umbrella is horizontally split at the waist, with the upper body incorrectly merged into the same cluster as the left-adjacent pedestrian. The road sign at the right-edge of the circle is segmented into three horizontal sections at its base, while the wall structure at the left-edge of the circle is vertically fragmented into multiple separate clusters. In the left circled region, the gas station fuel dispenser is longitudinally divided into four vertical segments, each further subdivided horizontally into upper and lower parts, surrounded by numerous small unmerged clusters.

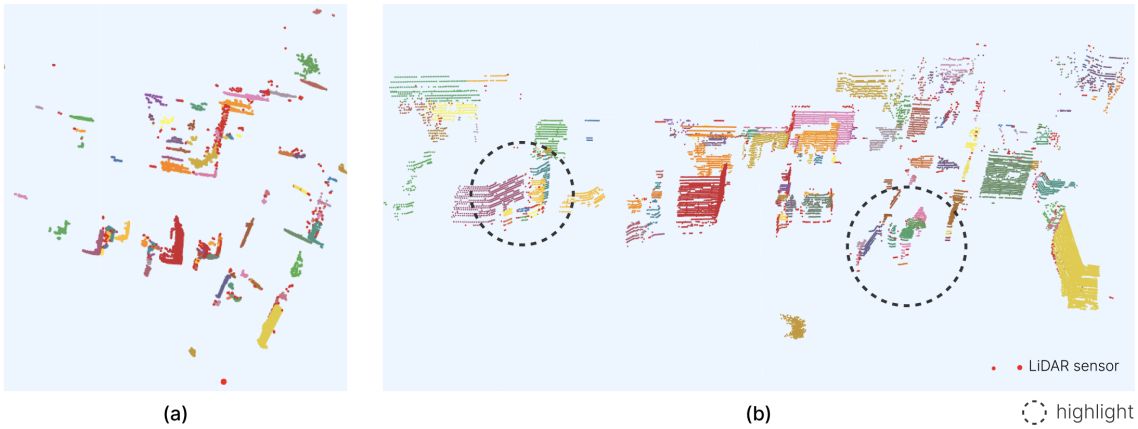


Figure 6.17: Clustering results in urban road section of Scenario 5 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

The RGAC and HBAC results present intriguing findings. Figure 6.18 part d compared to part c reveals a few more misclustered points on each object for the HBAC technique, as well as the tree in parts (g & h). However, a thought-provoking result can be found in parts g and h, where HBAC successfully clusters the pedestrians and even separates the umbrella from the pedestrian. Yet RGAC causes more under-segmentation because the person and the umbrella are clustered together. On the other hand, the opposite is happening in parts (e & f) since HBAC is grouping the fuel tank with the motorcycle (pink cluster) while the RGAC is separating them. Moreover, the circled car in (g & f) is split into two different clusters in both algorithms.

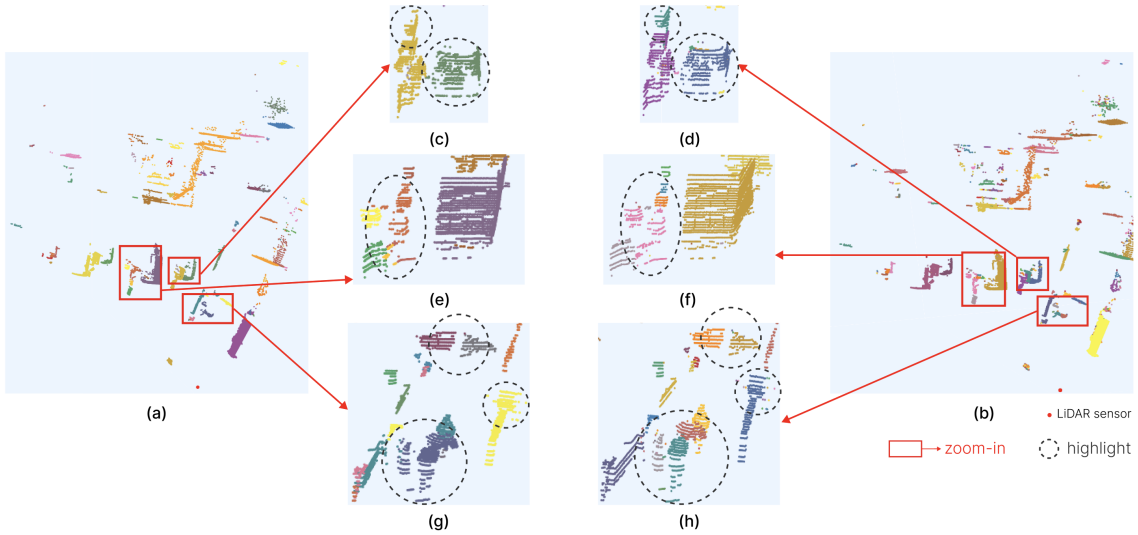


Figure 6.18: Clustering results in urban road section of Scenario 5 driving. (a),(c),(e),(g) Results using the RGAC algorithm. (b),(d),(f),(h) Results using the HBAC algorithm. (a),(b) Bird-eye view; (c)-(h) Zoomed-in key sections.

Scenario 6: Parallel Vehicle Configuration in Rain

The clustering results for scenario 6 are displayed in Figures 6.19, 6.20, and 6.21. The setting of scenario 6 presents multiple vehicles arranged in parallel lanes under rainy conditions, testing in dense traffic situations.

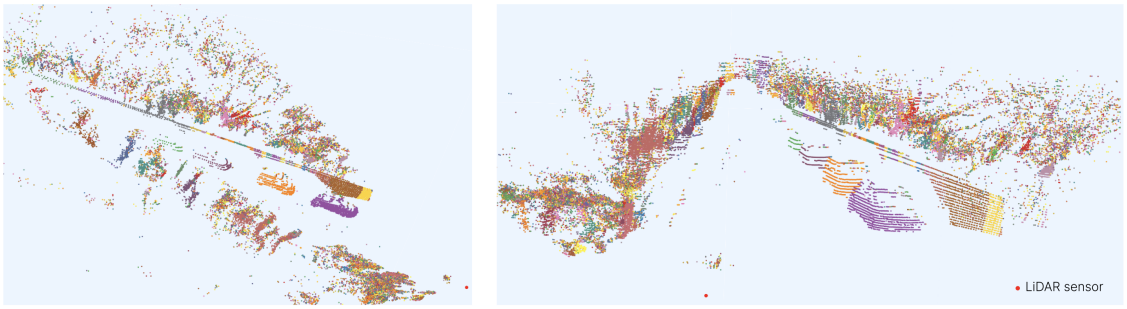


Figure 6.19: Clustering results in the rainy urban road section of Scenario 6 driving with baseline implementation. (a) Aerial oblique view; (b) Front view.

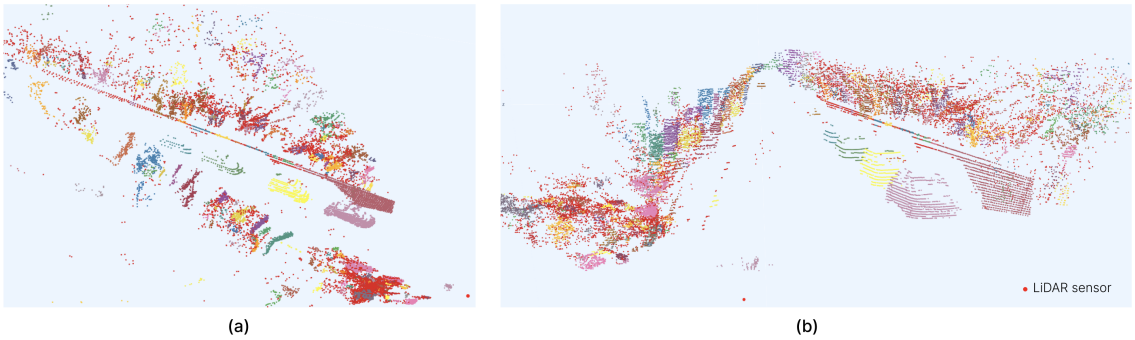


Figure 6.20: Clustering results in urban road section of Scenario 6 driving with HDBSCAN implementation. (a) Bird-eye view; (b) Front view.

HDBSCAN demonstrates relatively good performance in Figure 6.20 compared to previous rainy conditions. It successfully maintains most object boundaries without widespread over-segmentation, showing only outliers in small localized regions.

When comparing the proposed methods in Figure 6.21 (f & e), RGAC exhibits nearly perfect performance across all dynamic objects in the scene. It maintains precise cluster boundaries for vehicles in all positions. In contrast, HBAC shows slightly degraded performance with minor misclassifications evident in the highlighted circles across the front, middle, and rear sections of the scene. These areas reveal that a small number of points belonging to vehicles are misclustered. Despite these limitations, both proposed methods significantly outperform the baseline and HDBSCAN approach in maintaining coherent vehicle segmentation.

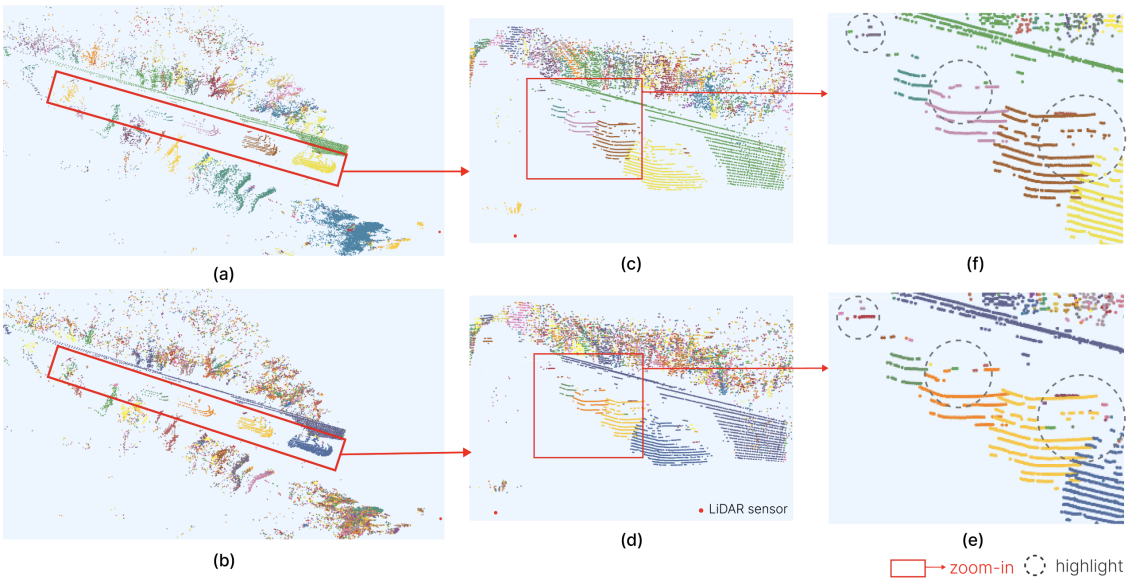


Figure 6.21: Clustering results in the rainy urban road section of Scenario 6 driving. (a),(c),(e) Results using the RGAC algorithm. (b),(d),(f) Results using the HBAC algorithm. (a),(b) Bird-eye view; (c)-(f) Zoomed-in key sections.

6.4.3 Summary of Visual Evaluation

In the standard urban scenarios (1-3), both proposed algorithms significantly outperform the baseline and HDBSCAN approaches, with RGAC showing better consistency in distant object clustering while HBAC demonstrates superior fine-grained separation. The rainy scenarios (4-6) reveal the robustness of both algorithms under adverse weather conditions, maintaining clustering performance despite atmospheric interference.

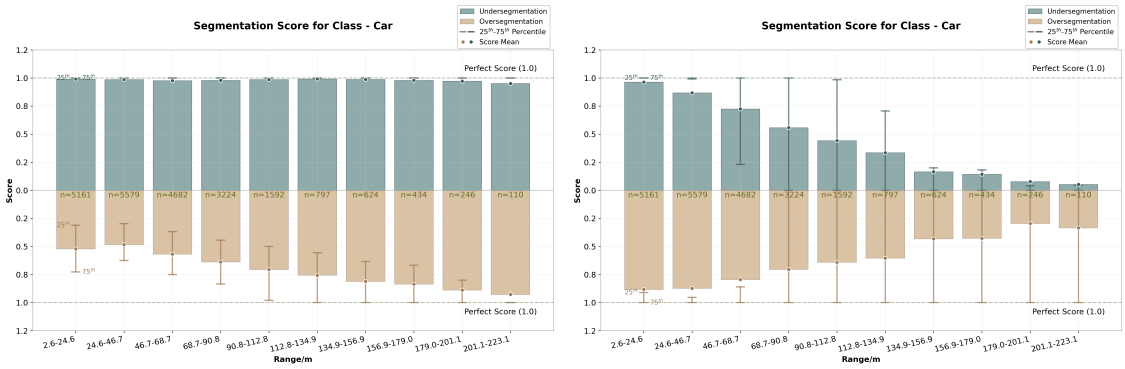
6.5 Quantitative Evaluation

In Section 5.2, the evaluation metrics are discussed (5.2.1) with a complete quantitative evaluation of the proposed algorithm. The histograms of the segmentation metrics performance for different object classes are within Figures 6.22-6.27, including car, truck, MC-moped, and pedestrian. Providing a performance comparison between the baseline, HDBSCAN, HBAC, and RGAC algorithms.

The histogram is designed with bidirectional bars, which are symmetrical up and down. The upper part of the histogram is represented by a blue bar chart showing the under-segmentation score, and the y-axis increases upward to '1.0'; With the y-axis growing downward to '1.0', the lower portion of a brown bar chart depicts the over-segmentation score; and '1.0' is the best segmentation score, indicated by the horizontal dotted lines in histograms. Each histogram has a set interval of 22 meters on its x-axis. Starting at 2.6m, it is progressively separated into intervals of 2.6 - 24.6m, 24.6 - 46.7m, etc. Different object classes set the x-axis extremum based on the actual detected maximum range. The vehicle category extends to a further away 223.1m in Figure 6.22.

Vehicle Class (Car and Truck): From close to distant range, the sample distribution shown by histograms drops off rapidly. The number of car samples in Figure 6.22 is 5,161 cases in the short range (2.6-24.6m), whereas 110 instances are in the maximum range (201.1-223.1m). The sample size is similarly decreased from 369 to 95 for the truck in Figure 6.23, so it is inherently constrained in scanning at a greater distance.

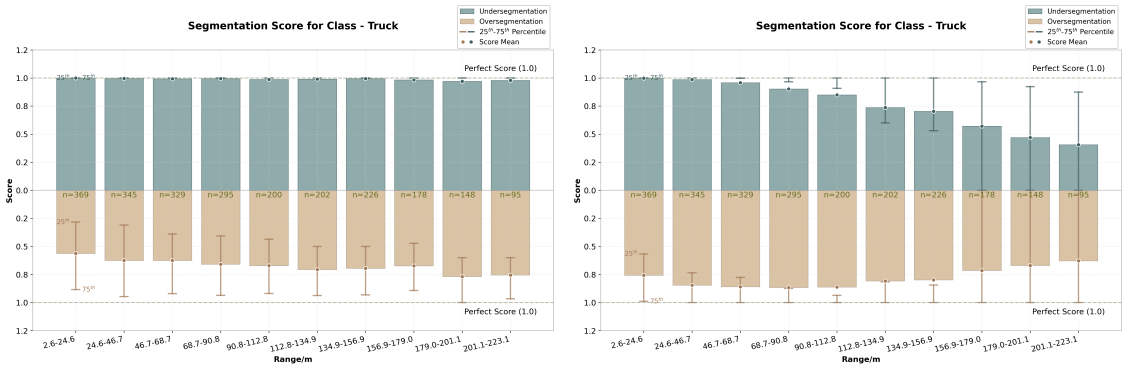
The baseline algorithm has a major over-segmentation issue, particularly within the medium range (2.6-112.8m), where the over-segmentation score for car is 0.5-0.8; The truck is 0.5-0.6, as shown in the Figures 6.22a and 6.23a. Performance uncertainty is seen by the greater span within the 25–75 percentile.



(a) baseline - class car

(b) HDBSCAN - class car

Figure 6.22: Evaluation histograms for class car: baseline & HDBSCAN.



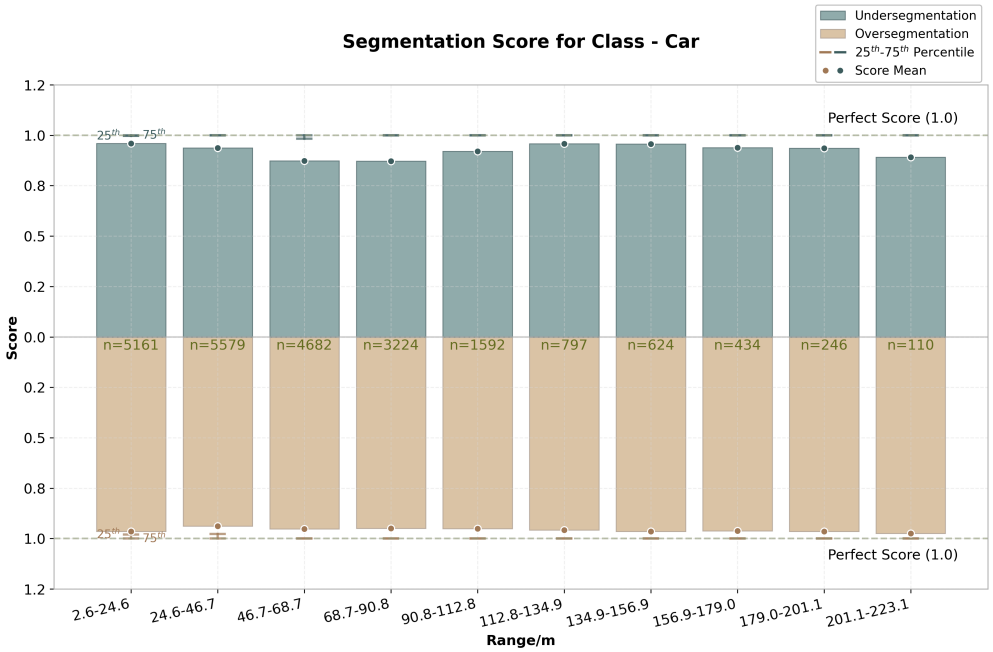
(a) baseline - class truck

(b) HDBSCAN - class truck

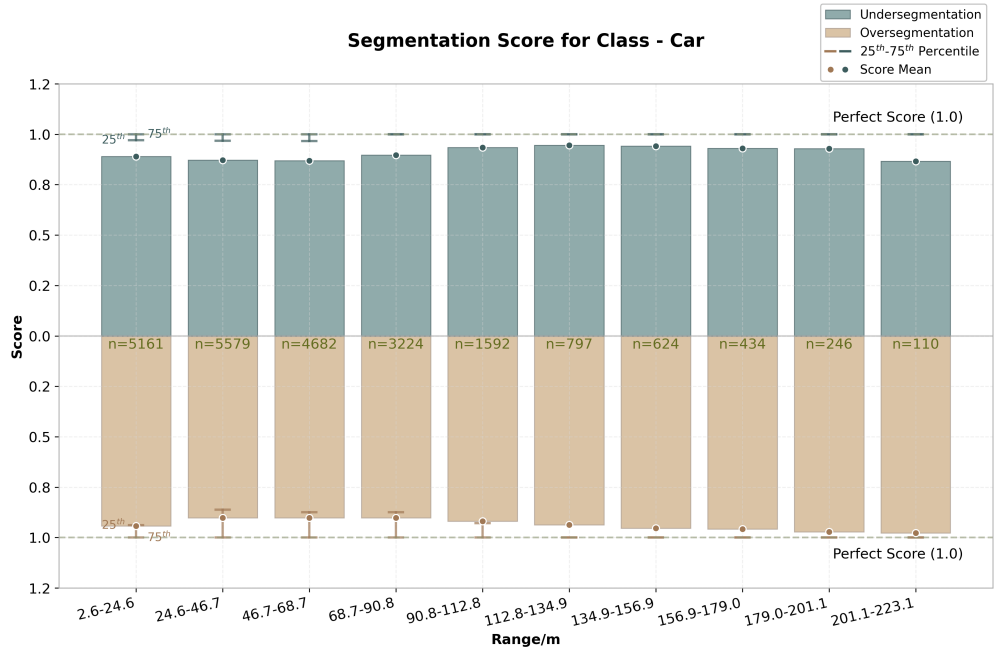
Figure 6.23: Evaluation histograms for class truck: baseline & HDBSCAN.

The HDBSCAN algorithm, with differing concepts, also has unique displays as shown in Figures 6.22b and 6.23b. The density-based clustering algorithm also shows a propensity for over-segmentation. Contrary to the baseline performance, the over-segmentation score lowers as the range increases. The score of cars drops from a maximum of 0.9 to 0.35-0.5; The trend for score of trucks is similar but relatively mild. The under-segmentation score shows a parallel pattern, as the range grows, the overall segmentation quality of the HDBSCAN algorithm decreases comprehensively, and it reflected in the 25-75 percentile with a large span at long ranges as well. Bidirectional decline of the algorithm performance is more in line with HDBSCAN's high reliance on point cloud density.

HBAC performance is better to the two methods mentioned before. The over-segmentation score consistently remains above 0.95 at all ranges, and the variance is significantly reduced in Figure 6.24b and Figure 6.25b. RGAC delivers the most stable over-segmentation performance, with a score close to 1 and the smallest variance within all ranges shown in Figure 6.24a and Figure 6.25a. However, RGAC has a unique behave in under-segmentation. The 25-75 percentile range of its under-segmentation score is close to 1, yet the mean deviates from and is lower than this range. This occurrence shows that the under-segmentation scores have a biased distribution; there are a few abnormal samples that perform with low scores, but the majority of samples perform well with scores near 1; The mean then falls as a result of extreme values.

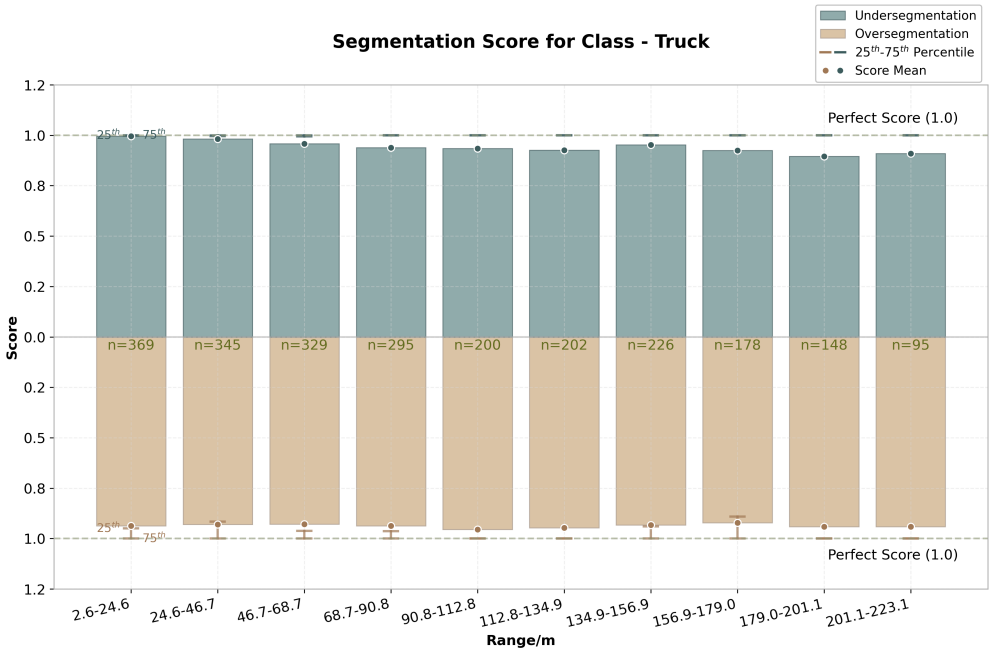


(a) RGAC - class car

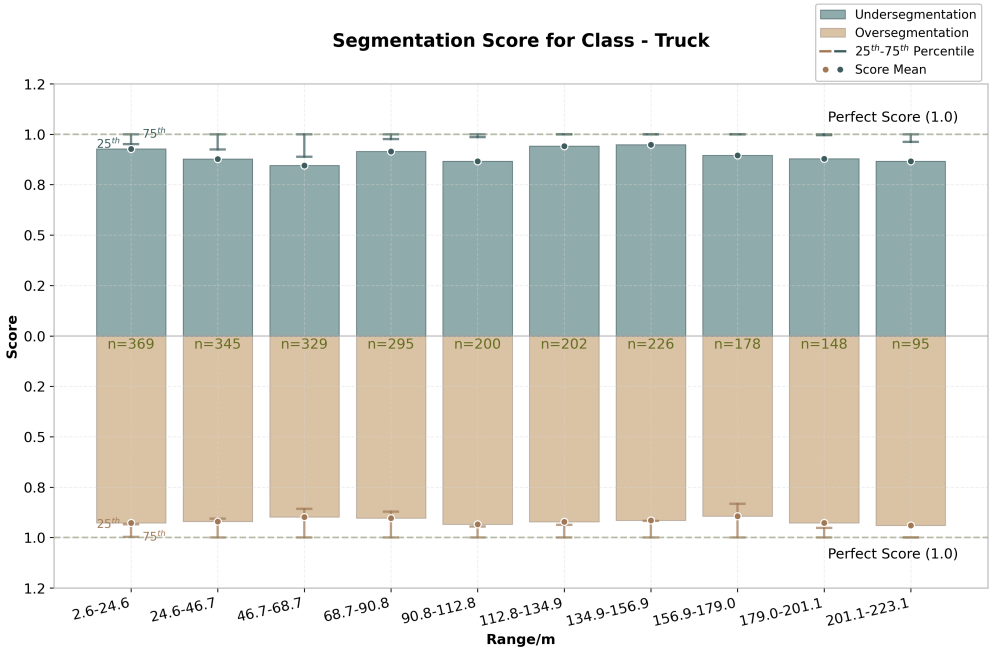


(b) HBAC - class car

Figure 6.24: Evaluation histograms for class car: RGAC & HBAC.



(a) RGAC - class truck

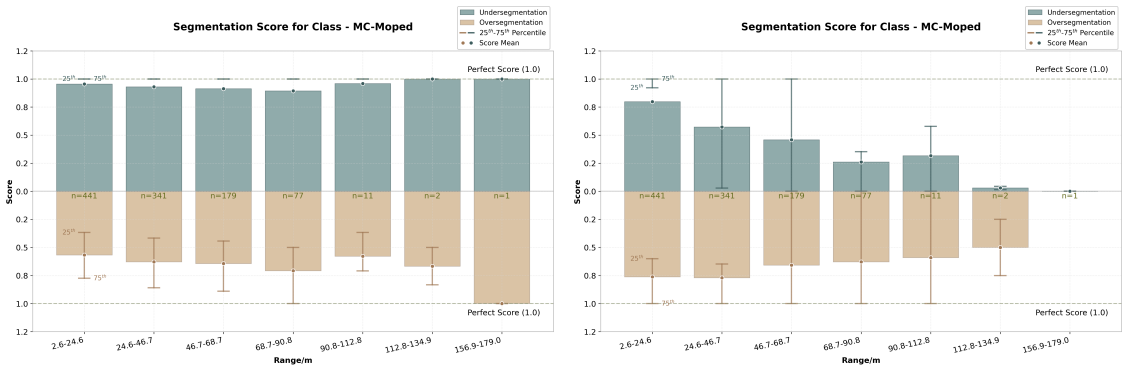


(b) HBAC - class truck

Figure 6.25: Evaluation histograms for class truck: RGAC & HBAC.

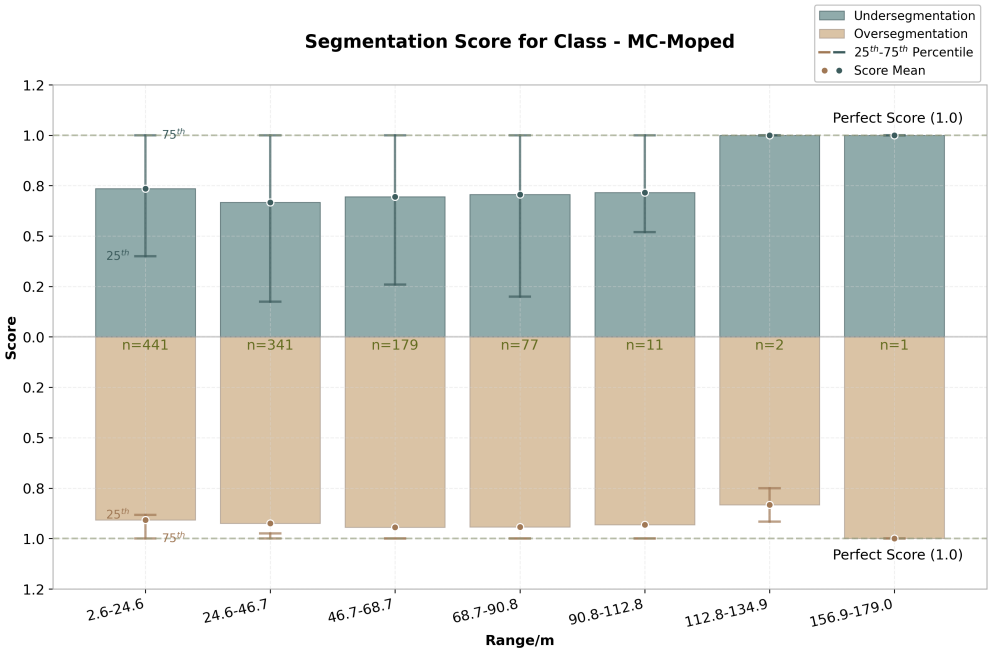
Motorcycle(MC)-Moped Class: The MC-Moped class performed in histograms with additional features, due to its small target size and sparse point cloud distribution. The sample size decreases vastly from 441 at close range (2.6-24.6m) to a single instance beyond 112.8 m. Unlike the larger vehicle classes (car and truck) which maintain big sample sizes across medium distances, MC-moped detection becomes statistically limited beyond 90m due to reduced LiDAR point density on small MC-moped vehicles.

The baseline algorithm demonstrates severe over-segmentation issues with scores reaching 0.5-0.6 across close-to-medium ranges, as shown in Figure 6.26a. HDBSCAN has the similar performance as discribed within 'Vehicle Class (Car and Truck)'. HBAC produces mixed results in Figure 6.26d. Over-segmentation improves to near-one levels in close-to-medium ranges but exhibits increased under-segmentation with the score of 0.55-0.7 and extremely high variance at the same time, suggesting difficulty in maintaining cluster coherence for sparse objects. This under-segmentation is more pronounced than observed in car and truck classes, likely due to the inherently sparse point clouds of small vehicles. RGAC achieves the most balanced performance in Figure 6.26c with consistently higher over-segmentation scores and moderate under-segmentation with 0.7-0.8, demonstrating superior handling of object clustering. The good performance across all three algorithms for MC-moped at longer ranges, where both over- and under-segmentation approach 1.0; yet it is compromised by extremely limited sample sizes of only 1-2, weakening the statistical representativeness for meaningful analysis.

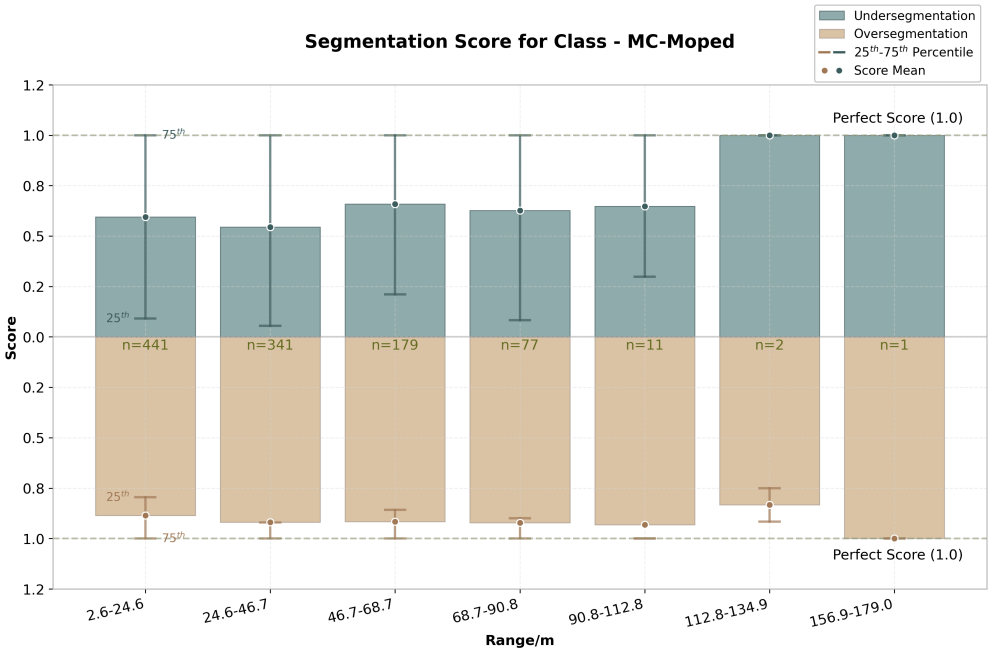


(a) baseline - class MC-moped (b) HDBSCAN - class MC-moped

Figure 6.26: Evaluation histograms for class MC-moped.



(c) RGAC - class MC-moped

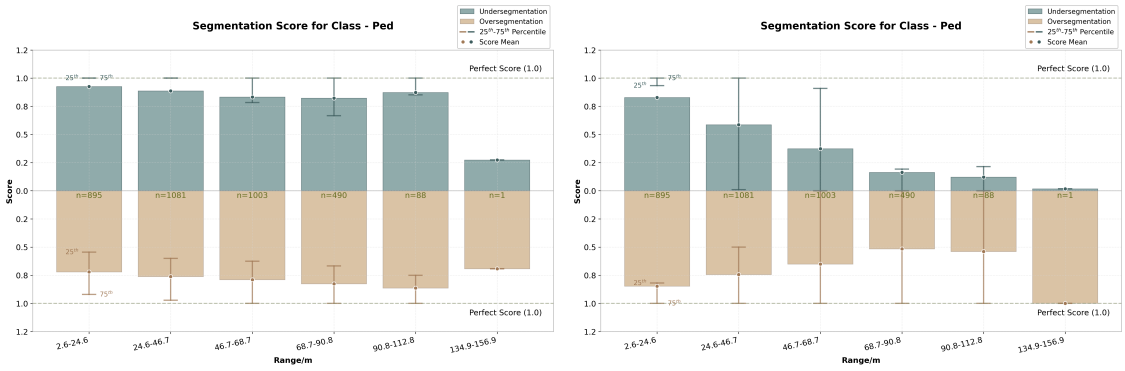


(d) HBAC - class MC-moped

Figure 6.26: Evaluation histograms for class MC-moped.

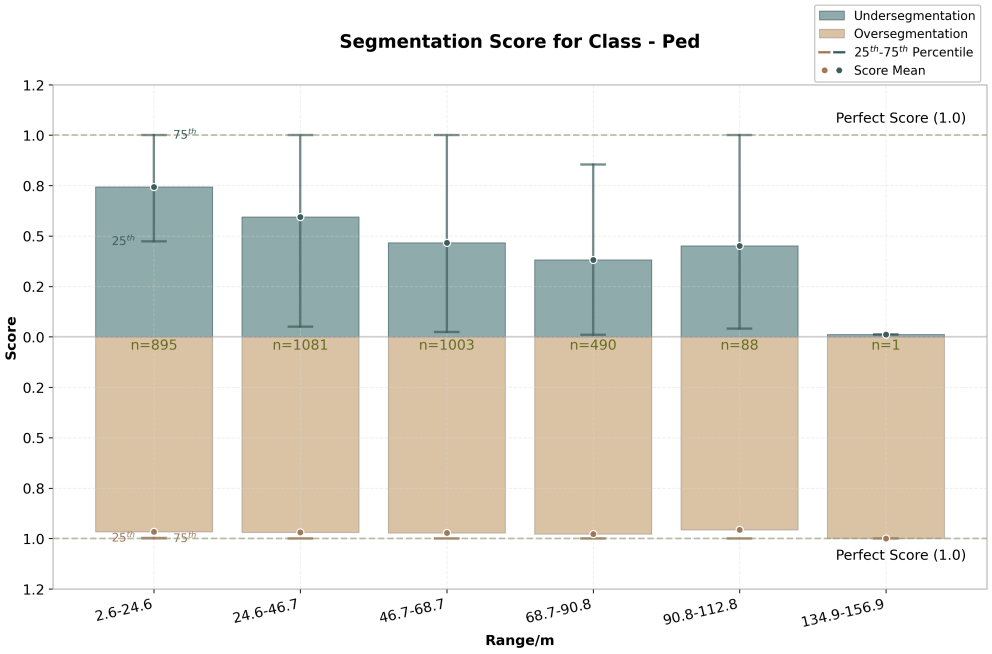
Pedestrian Class: The pedestrian class shows completely different segmentation performance from the motor vehicle classes above. The sample size distribution, with a peak in the medium range (24.6-68.7m), has more than 1,000 instances rather than the short range. This contrasts with the vehicle classes display where the number of samplesshrinks with range. It reflects that in the urban environment, pedestrians at close range may be partially blocked; Or the sensor can capture pedestrians well within the range within an ideal ranges.

The over-segmentation performance demonstrated by the baseline algorithm is 0.7-0.8 across the full range as shown in Figure 6.27a, which is superior to 0.5-0.6 observed in the vehicle classes. Perhaps due to the larger and more complex geometry compared to vehicles, pedestrians have a compact spatial distribution feature. HDBSCAN has a similar performance as described for other classes. The performance of HBAC is further improved as shown in the Figure 6.27d. Successfully obtain an over-segmentation score close to 1.0 within all ranges; However at the same time, the undersegmentation score is low, ranging from 0.4 to 0.6, and the variance is large, indicating the existence of inconsistency. RGAC achieves the same ideal performance in Figure 6.27c, with consistently high over-segmentation scores. While the under-segmentation matrices improve to around 0.8 within a near range, the overall over-segmentation performance is rather inadequate. Similar to the MC-moped motorcycle class, with no superiority of HBAC and RGAC in the maximum range due to insufficient sample size, thus lacking statistical significance.

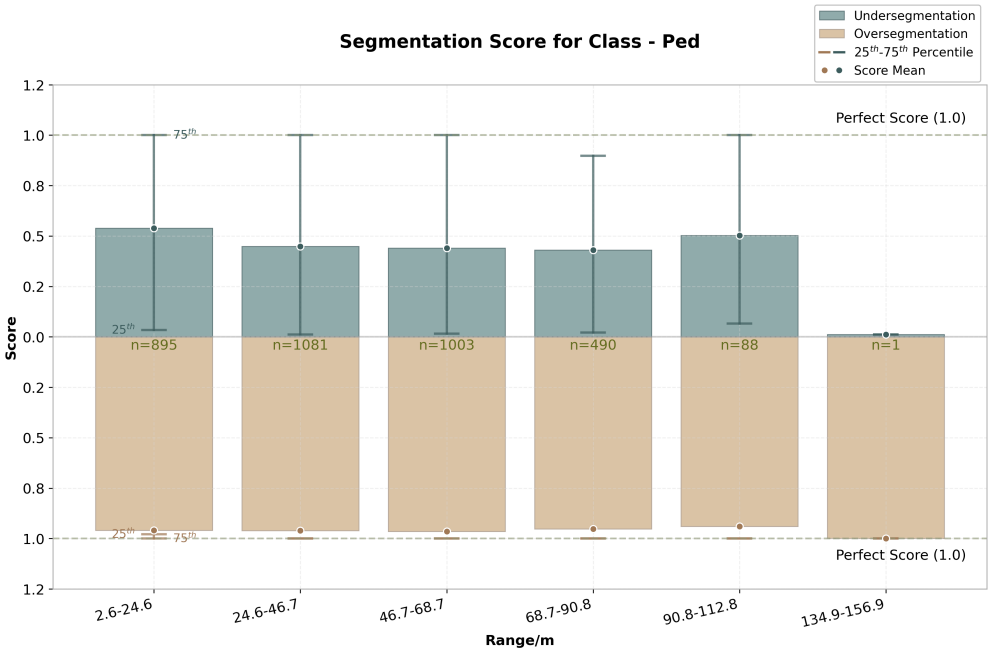


(a) baseline - class pedestrian (b) HDBSCAN - class pedestrian

Figure 6.27: Evaluation histograms for class pedestrian.



(c) RGAC - class pedestrian

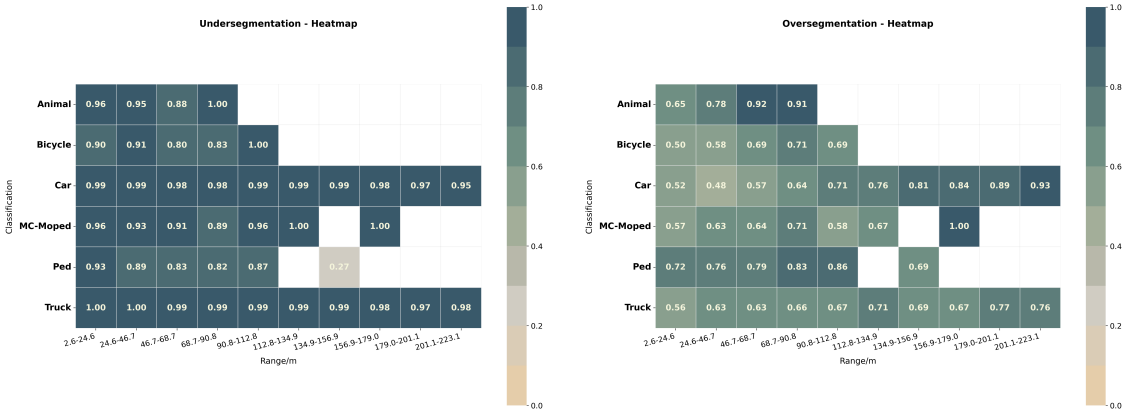


(d) HBAC - class pedestrian

Figure 6.27: Evaluation histograms for class pedestrian.

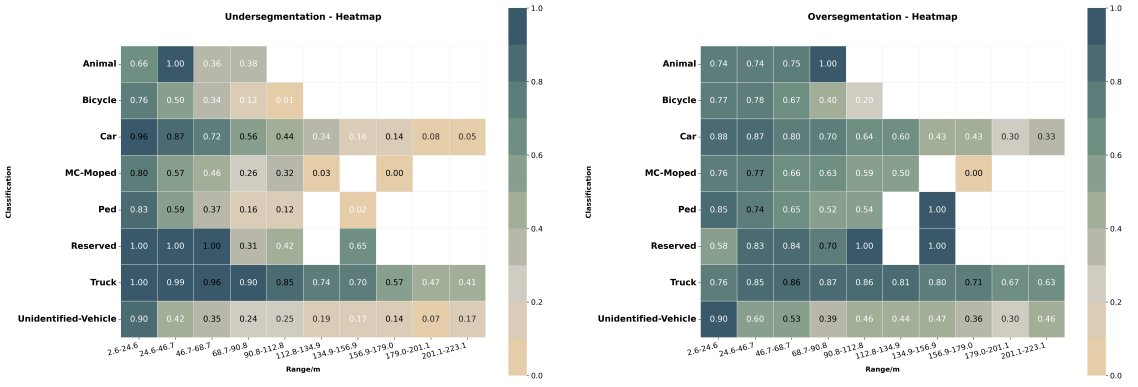
Comprehensive Heatmap Analysis: To present a more comprehensive evaluation, the quantitative performance of all algorithms, categories and distance ranges is demonstrated through heat maps. This analysis includes additional target categories beyond the categories of cars, trucks, MC-mopeds and pedestrians analyzed in the histogram, which are animals and bicycles (the histogram performance results of these two classes enclosed in Appendix A.1). In the heat map, the horizontal axis represents the range, while the vertical axis represents different target categories; Color intensity indicates the level of the segmentation score; the darker the color, the better the performance.

Figures 6.28a and 6.28b respectively present the heat maps of the undersegmented and over-segmented performance of the baseline model. Similarly, Figures 6.29a and 6.29b are the illustration of HDBSCAN. Figures 6.31a and 6.31b show the corresponding performance indicators of the HBAC algorithm, while the RGAC algorithm's are displayed in Figures 6.30a and 6.30b.



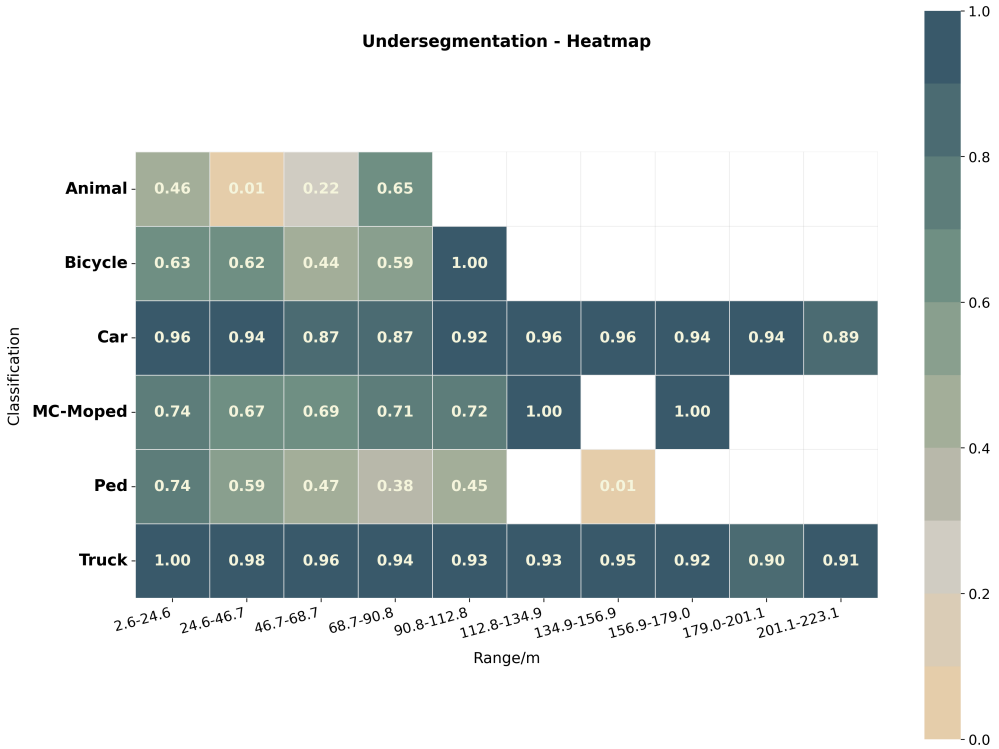
(a) under segmentation score (b) over segmentation score

Figure 6.28: Evaluation heatmaps for baseline algorithm.

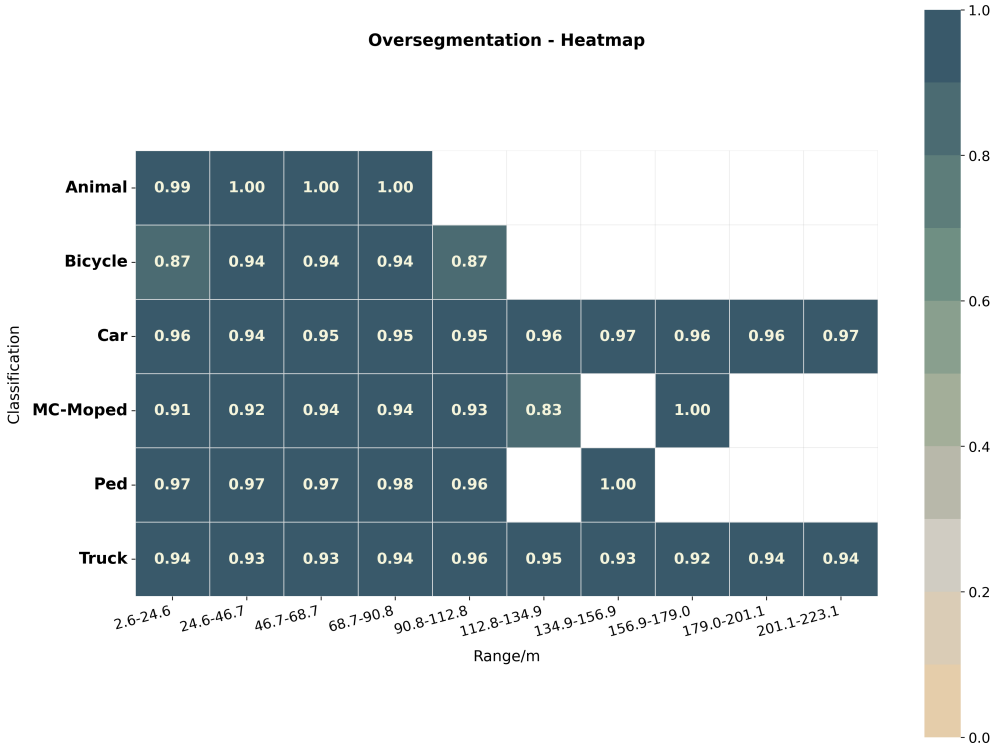


(a) under segmentation score (b) over segmentation score

Figure 6.29: Evaluation heatmaps for HDBSCAN algorithm.

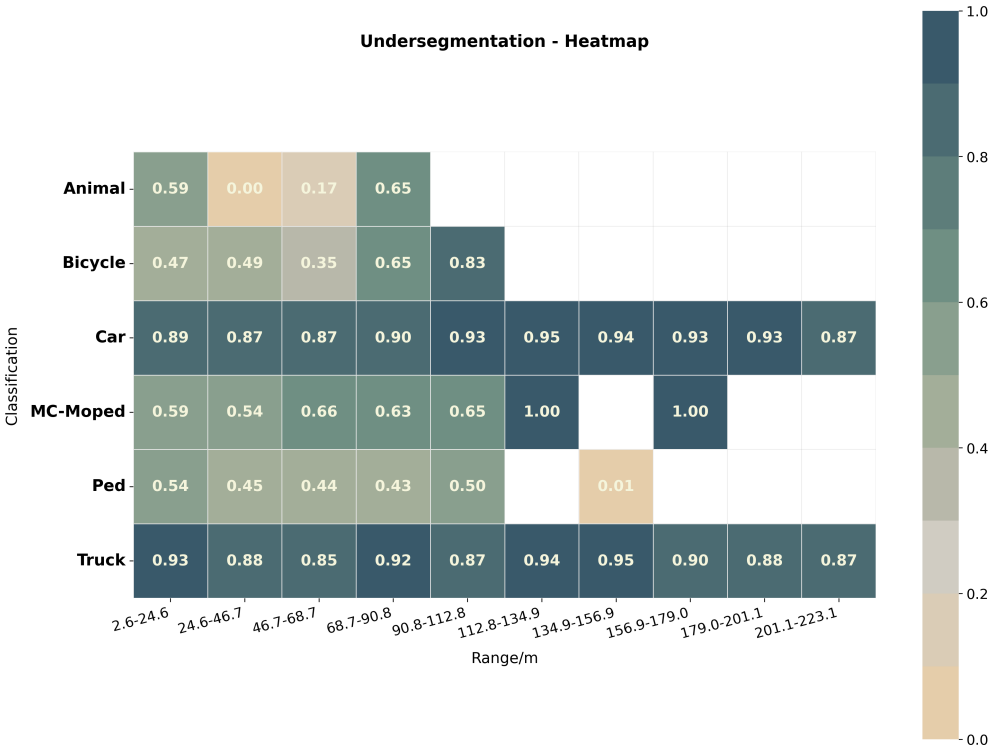


(a) under segmentation score

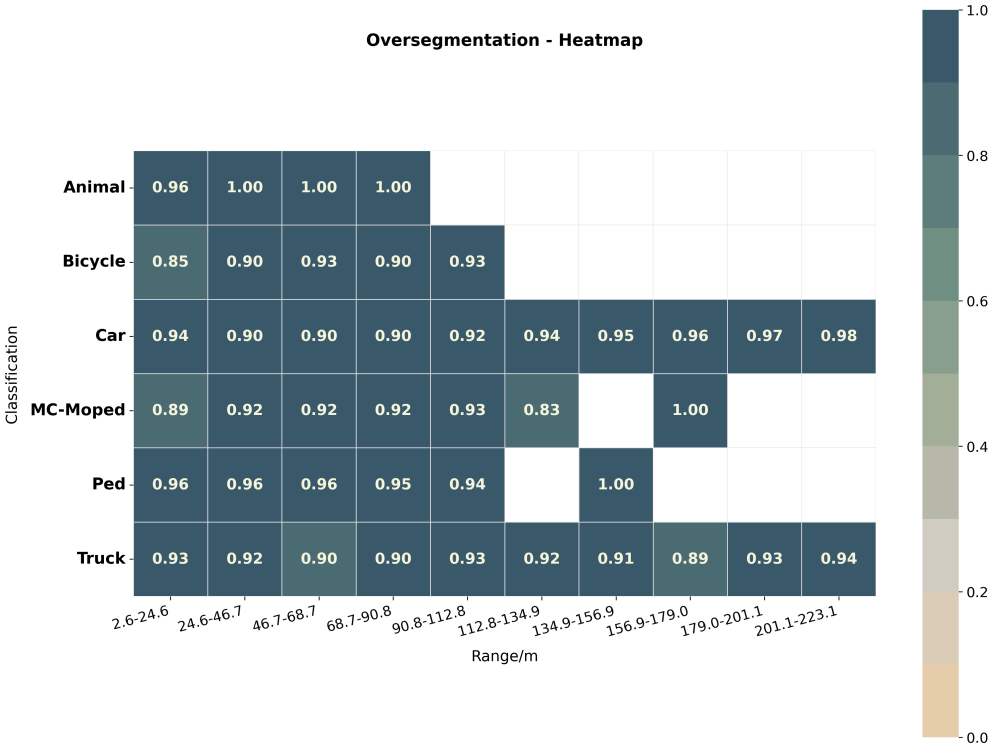


(b) over segmentation score

Figure 6.30: Evaluation heatmaps for RGAC algorithm.



(a) under segmentation score



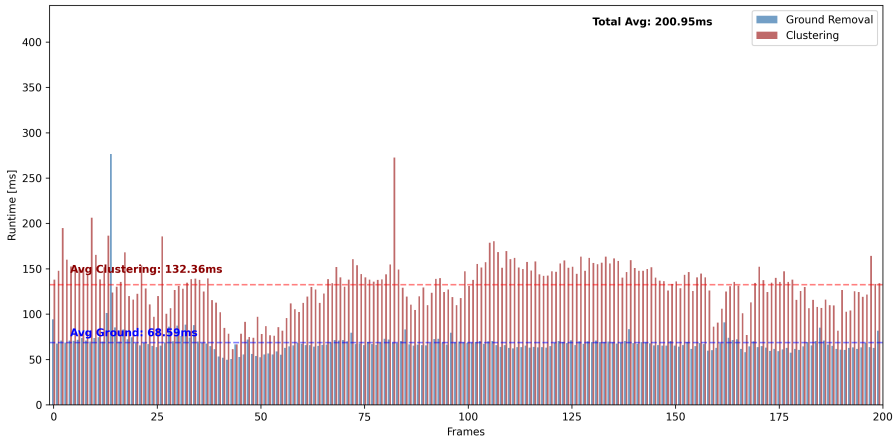
(b) over segmentation score

Figure 6.31: Evaluation heatmaps for HBAC algorithm.

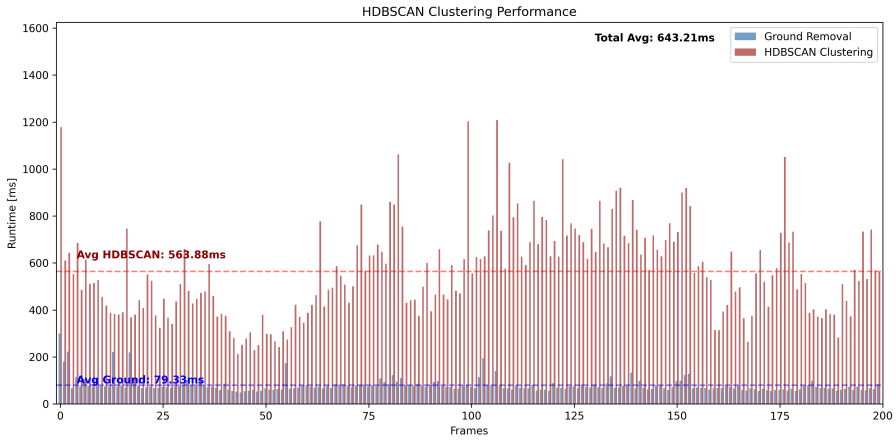
6.6 Computation Time

The computational efficiency of all the proposed algorithms is considered. The experiments are conducted on a workstation equipped with an Intel(R) Core(TM) i9-10885H CPU@2.40GHz processor. The RGAC and HBAC algorithms are developed using Python; As for the baseline algorithm, the original C++ model is modified to use the Python programming version in order to conduct a fair comparison of the timing. Future optimization may be feasible as all calculations are done on the CPU without the need of GPU acceleration technologies like CUDA for this step.

Figure 6.32 shows the processing time of all the algorithms deployed on 200 frames of LiDAR data. The baseline algorithm (Figure 6.32a) achieves the fastest running time, with a total average time of 200.95ms. Nevertheless, the HDBSCAN performance (Figure 6.32b) has the highest computational cost. The total processing time is 643.21 ms. It is mainly due to the density-based clustering method's high neighborhood computation requirements.

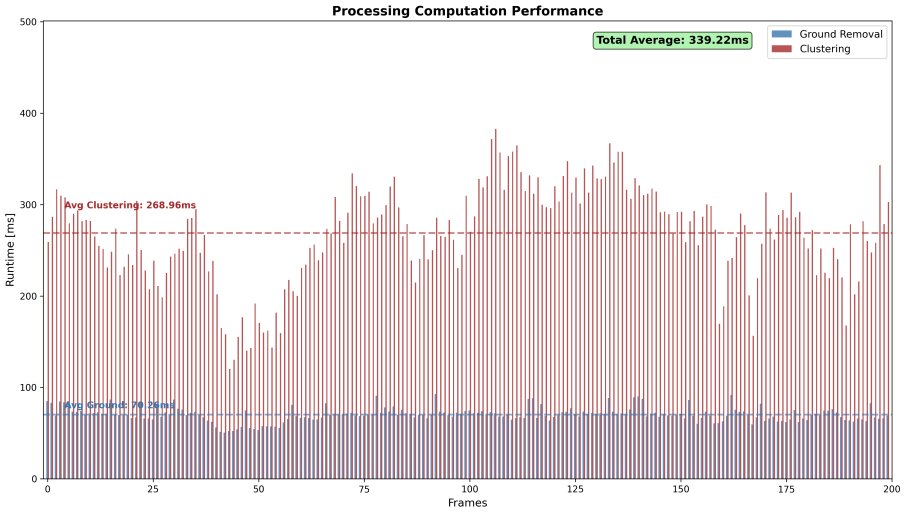


(a) baseline Ground Removal & Clustering

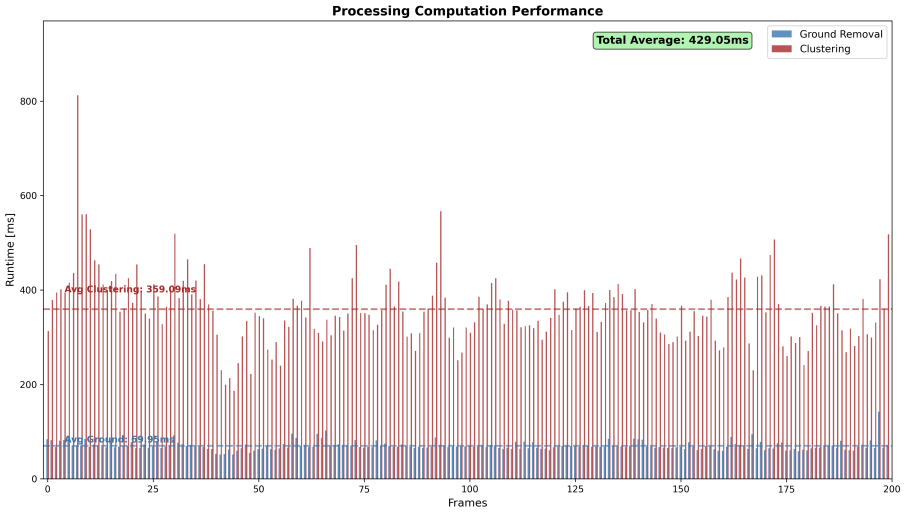


(b) HDBSCAN Ground Removal & Clustering

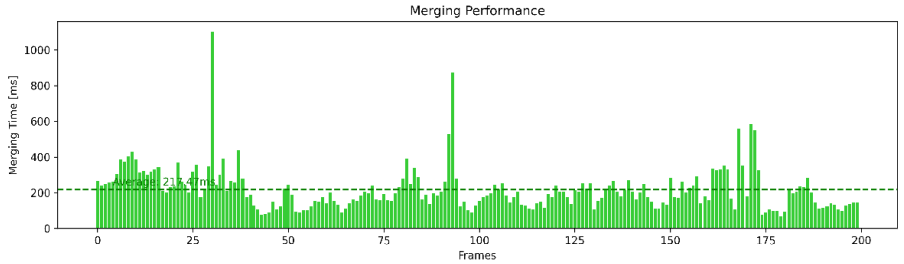
Figure 6.32: Evaluation of computation time.



(c) RGAC Ground Removal & Clustering



(d) HBAC Ground Removal & Clustering



(e) Merging Computation

Figure 6.32: Evaluation of computation time.

HBAC (Figure 6.32c) and RGAC (Figure 6.32d) show intermediate computational costs with their advanced range-image processing techniques. Figure 6.32e illustrates the additional merging computation time used.

Table 6.1: Average computation time (ms) comparison across algorithms

Algorithm	**Ground Removal**	**Clustering**	**Total**
Baseline	68.59	132.36	200.95
HDBSCAN	79.33	563.88	643.21
RGAC	70.26	268.96	339.22
HBAC	69.95	359.05	429.00

The results show that HDBSCAN requires the longest clustering time due to its computational density. The two proposed algorithms show temporal variations between different frames. And occasional temporal peaks can be observed in the clustering stage. The merge calculation (Figure 6.32e) adds an average time cost of 217.47 milliseconds for processing optimization.

7

Discussion

In this chapter, the results highlighted in the Chapter 6 will be discussed. Firstly, the performance of both ground removal methods will be examined, followed by an analysis of why the Baseline approach performs poorly. Furthermore, there will be a discussion about the two developed approaches, RGAC and HBAC, including their performance, requirements, limitations, and a comparison to determine which is better. Moreover, there will be an evaluation of the reliability of the results shown in Chapter 6. Lastly, the Chapter will end with computation time analysis and recommendations for future work.

7.1 Ground Removal Performance Analysis

Ground removal experiments within data processing were conducted in the complex urban environment. The results show that the dual-grid adaptive method performs better than the traditional methods.

Basic ground removal methods assume that the ground is level, which does not work well in real situations. The results show that the urban environment has ground at different heights, including roads, sidewalks and pedestrian paths etc. None of these can be uniformly removed merely through simple height restrictions. By employing two grid sizes, the dual-grid system addresses the up-and-down of the road surface. The large grid is used to obtain the overall ground conditions, and the small grid is used to address the changes at the junction of the target and the ground. So with obtaining the distribution of various ground types, the method helps accurately identify the target edge as well.

The main improvement of the method is to adjust its height limit according to the local ground conditions that meet conditions. In areas with significant height variations, where dynamic objects may appear, the algorithm employs stricter restrictions; while in flat areas, it efficiently only imposes normal restrictions. Besides preventing the removal of lower-located targets, it also removes the points surrounding independent targets that are connected to the ground.

The quality of ground removal directly affects how well clustering works. As shown in Figure 6.1, poor ground removal causes problems where different objects are grouped because ground points connect them. By removing these improper connections while maintaining the intended boundaries, the dual-grid approach establishes a good base

for clustering in the next step.

7.2 Baseline Approach

The baseline that was chosen was explained Section 3.4.10. It is an attractive approach, since it is fast and simple, since it only requires one parameter or threshold $\theta_{Baseline}$. However, the results of the non uniform LiDAR data shows the opposite since the performance of the algorithm was poor with many clustering errors. There are two reasons for the poor performance.

The first possible reason is that the algorithm requires a sensor with fixed angular resolution in the elevation direction, which this project does not have. Instead, the angular resolution in the vertical direction was set to as in HBAC $\alpha_v = |p.\phi - q.\phi|$. This results in instability since the trigonometric relationship that determines the β angle is also dependent on α_v. The other reason it performs badly, is that the baseline approach only has one threshold $\theta_{Baseline}$ for both vertical and horizontal direction, which will result in extra over/under-segmentation if the angular resolution is not the same in both direction. This is also magnified because of the non-uniformity in the elevation axis.

7.3 K-D Tree Merging Step

To ensure that the outliers being merged are most likely due to the issues described in Section 4.2.2 and not the clustering algorithm's performance, the predetermined maximum size of the cluster to be taken into consideration for this merging phase is set to the value three, as clusters of size one to three showed similar performance, and increasing the threshold beyond three had no noticeable impact. A faulty clustering algorithm would typically produce larger misclustered regions, not isolated small clusters. Consequently, the results of the clustering algorithms keep their reliability while minimizing the effect of the problems of the missing measurements.

Furthermore, the merging step is only applied to outliers if they lie near a single dominant (i.e., largest cluster in the vicinity of the outlier) cluster. Thus, avoiding the creation of new large clusters in regions with many scattered points, such as vegetation. These scattered points appear as isolated points in the range image (i.e., in areas with many missing points).

An example is in Figure 6.2, where there is a point on top of the car in part d) which does not have any vertical or horizontal neighbors in the range image from the car itself, because it is isolated. Instead, the neighbors in the range image correspond to points farther away in the background where the vegetation is present as shown in Figure 5.1a, resulting in it being clustered incorrectly. Nonetheless, the K-D tree merging step is based on Euclidean distance rather than the range image. Consequently, such outlier points are reassigned correctly if they fall within the merging radius

7.4 Parameter Tuning

One of the major limitations of model-based clustering algorithms (including the proposed approaches) is that they are highly affected by the value of a set of parameters rather than learning from data, as in machine learning. This adds a layer of complexity when tuning the parameters which can be time consuming since there is a lot of trial and error. However, the advantages of such model-based approaches are that they are more controllable and interpretable. In this thesis, the parameters are tuned empirically based on the visual inspection of the clustering results and the over and under-segmentation scores by running the algorithm repeatedly. Lastly, this process is subjective because it demands trade-offs between over- and under-segmentation, which makes it hard to generalise.

7.5 RGAC vs HBAC

The two developed approaches had comparable results, where they both outperformed the baseline approach on every scenario of the visual evaluation. One pattern that can be seen from the performance difference between RGAC and HBAC is that HBAC has tendency to result in slightly more over-segmentation errors while RGAC is relatively more stable.

The exact cause of this difference is uncertain. A possibility can be the problems discussed in Section 4.2.2. However, that is unlikely since similar issues to the same extent should be visible in the RGAC results, which they are not, because both methods handle these problems in similar ways, such as the neighbor exploration, gap handling, and the range difference solution for the partial occlusion. This suggests that the problem is in the HBAC clustering mechanism.

However, it is not straightforward to identify the exact cause of the problems present in HBAC since debugging the problem is hard. Another observation the can be drawn from the results of HBAC is that there is over-segmentation in the vertical direction. This occurs when points on top of each other are assigned to different clusters, or when an entire layer is clustered differently from the adjacent layers.

A strong indicator of the HBAC problem is the circled distant car in Figures 6.6.c and 6.6.d appear to be well clustered in RGAC, but not in the HBAC, which fails to cluster it vertically, while horizontally (per layer) it is relatively stable. Another example of this phenomenon is seen in the results of Figures 6.9.d and 6.12.f for the furthest circled car, where a vertical layer is being misclustered. This is also true for the results of Figure 6.15 and Figure 6.21.

This can be attributed to the non-uniform angular resolution α_v in the elevation direction which results in inconsistent vertical thresholding in the ABD criteria. Another possible explanation is that there is another layer of uncertainty in the adaption of the ABD clustering step in Equation 4.4 since it is an approximation of the non-uniformity.

Lastly, partial occlusions remain a challenge. In Figure 6.12.c and 6.12.d, in Figure

6.19.g and 6.19.h, both the bus and the circled distant car are clustered into two different clusters. There are two possible reasons for this problem: first, the gap between the disconnected object parts (e.g., a car or a bus) exceeds the maximum steps max_{steps} in the gap handling solution in HBAC and RGAC. Second, the clustering conditions, whether it is DBD or ABD are not fulfilled due to the parameter tuning. Adjusting these parameters could merge the disconnected parts, but it can also introduce under-segmentation with the other clusters.

7.5.1 Quantitative Evaluation

As seen in Section 6.5 both HBAC and RGAC achieved a better over-segmentation score in comparison to the baseline, while the baseline performed better on the under-segmentation score. The relationship between over-segmentation errors and under-segmentation errors is strictly inverse, since one object can be both over-segmented and under-segmented simultaneously. Nonetheless, there remains a risk that while avoiding over-segmentation, the algorithm may merge distinct objects (especially at a distance), leading to under-segmentation as explained in section 4.2.1. This explains why RGAC and HBAC perform better on over-segmentation and slightly worse on under-segmentation. Furthermore, HBAC had a slightly worse over-segmentation score than RGAC, which is attributed to the reasons discussed in the parent section. However, the reason for the decreased under-segmentation performance turned out to be that the angular check of DBD is merging points together, which initially were not clustered by the ABD check. It turned out that the fixed angular threshold in RGAC is a better approach, since it is both computationally lighter and performs better.

The proposed HBAC and RGAC algorithms resulted in a balanced and robust performance compared to the baseline model across different categories (e.g., Cars, Trucks, Bikes, etc) and ranges. This superior performance highlights the strength of the range image based clustering techniques.

7.6 HDBSCAN

HDBSCAN does not depend on the range image, and it works based on the density of the point cloud; thus, all the problems related to missing points are not visible. However, the disadvantage of density-based methods is that when the point cloud is non-uniform, there is the possibility that the gap between object points is non-uniform, and the object will be over segmented.

HDBSCAN works directly on the 3D point cloud without relying on a range image representation, which makes it naturally robust to the problems of range image clustering, such as the Angle-Based Clustering and the missing measurements problems. This is because clustering is based purely on spatial density rather than positional indexing. As a result, there is no need for the merging step. Furthermore, the clustering results from HDSBCAN seem to successfully cluster partial occlusion situations as seen in the bus in Figure 6.11.

Interestingly, this strength is also where HDBSCAN's main limitation lies, because the clustering is happening based on the density of the point cloud. This non-uniformity may cause points from the same object to have different densities and thus be clustered differently, as seen in the tree trunk in Figure 6.5 and the truck in Figure 6.14.

7.7 Computation Time Analysis

The baseline algorithm in [10] is quite fast since it operates directly on the range image rather than expensively processing the 3D point clouds. The algorithm processes each cell in the range image using a single-pass BFS approach. Furthermore, it only requires basic trigonometric calculations per cell. Thus, the order of the algorithm is O(n). The order of the two developed approaches, HBAC and RGAC, is still O(n), but with some computational overhead that approximately doubles the computation time, as highlighted in the results in Section 6.6.

In HBAC, the ABD requires trigonometric calculations. Additional complexity is added in the DBD by including a vector angle comparison between triplets of points. Additionally, in HBAC there are gap handling and partial occlusion difference checks on lines 9-13 in Algorithm 4, which involves evaluating two neighboring points. All these steps add extra computational overhead. Despite that, all these operations are O(1). Furthermore, max_{steps} prevents search operations from scaling with input size, which results in the algorithm being O(n) which is the same complexity as the baseline. However, it will be slower than the baseline approach because of the heavier trigonometric calculations.

On the other hand, the speed performance of the RGAC is better by approximately 15-30% while keeping the complexity of O(n). The computation time of the RGAC is better than HBAC due to elimination of the expensive DBD angle checks, reduction of the gap searches to only one neighbor, and the replacement of the vertical ABD with fixed range thresholds.

The computation results of the baseline paper [10], where the code was written in an optimized, efficient language (C++) suitable for real-time applications, the baseline algorithm achieved an average runtime of 4.7 ms. While the baseline implementation in Python achieved an average runtime of 132.32 ms. This can give us a sense of how quickly the proposed algorithms would have performed if they were written in an efficient language as C++. Thus, RGAC performed almost twice as slowly as the baseline but still over 90 ms quicker than the HBAC approach. Consequently, this indicates that even though the developed approaches, which have better cluster performance, are slower than the original baseline, they are still considered to be quick if implemented in C++.

As for the K-D merging algorithm, it has average-case time complexity of O(n log n). Furthermore, the runtime increases moderately with the number of processed points. However, the performance is also highly dependent on the structure of the point cloud, which changes based on the number of point cloud detections per scene. If there are a lot of missing points, which will result in more isolated individual points,

this step could slow down to $O(n^2)$ time in the worst case scenario. This additional complexity raises the question of its necessity, especially since the result only merges a few outliers if they are present. If precision is important, then using the merging function will slightly improve accuracy, but this step is not required if speed is more important.

Finally, the HDBSCAN had the worst computation time of 564 ms as expected. This is because HDBSCAN is hierarchical density-based clustering, which typically requires calculating pairwise distances and constructing a minimum spanning tree, leading to higher computational complexity (often above O(n log n)).

7.8 Future Work

Future work includes better gap handling and partial occlusion techniques than our naive approach. One solution to that problem is using Map connections introduced [25], which is an enhancement to the standard range image representation of LiDAR data which connects non-adjacent cells in the range image. The map connections were calculated using known angular offsets and are applied at specific intervals (e.g., every second or third point) to bridge missing or obstructed measurements. These connections maintain the object surface continuity even when the data is sparse or partially occluded. This will improve the clustering performance but also eliminate the need to do post post-processing step with K-D tree, thus increasing computational efficiency.

Another interesting area to investigate is GPU accelerated HDBSCAN. By working with HDBSCAN the missing points problem will not be relevant since it is not range image based. However, to mitigate the high computation time of HDSBCAN, it is recommended to accelerate it on a GPU. As a result a near real time performance might be accomplished.

The final improvement area for this project is investigating different automation approaches to the tuning of the parameter process, as this will result in huge time saved during the development phase.

8

Conclusion

In this thesis, two model-based clustering algorithms were developed, HBAC and RGAC, extending the classical break point and range image clustering methods to account for non-uniform point cloud distribution.

In addition to proposing these algorithms, the thesis introduced two complementary ground removal methods and outlined both visual and quantitative evaluation strategies. A private dataset of Luminar point clouds was used to assess performance, and the results demonstrated the effectiveness of the proposed approaches over a baseline method. However, the approaches kept relative computational efficiency in comparison baseline method since all three operate with O(n) complexity.

Unlike deep learning systems, the model-based methods given here do not require large-scale labeled datasets, making them more adaptable to new scenarios.

Lastly, future work is to propose better gap handling and partial occlusion techniques than our naive approach. This will improve the clustering performance but also eliminate the need to do post post-processing step with K-D tree, thus increasing computational efficiency. Furthermore, investigating GPU accelerated HDBSCAN can be a promising field.

References

[1] Q. A. Abdullah, "The layman's perspective on technical theory and practical applications of mapping and gis," 2014.

[2] M. Ackerman and S. Ben-David, "A characterization of linkage-based hierarchical clustering," *Journal of Machine Learning Research*, vol. 17, no. 231, pp. 1–17, 2016.

[3] M. Adnan, G. Slavic, D. Martin Gomez, L. Marcenaro, and C. Regazzoni, "Systematic and comprehensive review of clustering and multi-target tracking techniques for lidar point clouds in autonomous driving applications," *Sensors*, vol. 23, no. 13, p. 6119, 2023.

[4] M.-I. Akodjènou-Jeannin, K. Salamatian, and P. Gallinari, "Flexible grid-based clustering," in *Knowledge Discovery in Databases: PKDD 2007*, J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 350–357, ISBN: 978-3-540-74976-9.

[5] S.-Y. An, J.-G. Kang, L.-K. Lee, and S.-Y. Oh, "Line segment-based indoor mapping with salient line feature extraction," *Advanced Robotics*, vol. 26, no. 5-6, pp. 437–460, 2012.

[6] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[7] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[8] T. Ayre, *Point Cloud Insights: Delving into Applications & Importance — ths-concepts.co.uk*, Accessed 04-05-2025, 2024. [Online]. Available: `https://www.ths-concepts.co.uk/what-is-a-point-cloud/#:~:text=An%20unordered%20point%20cloud%20is,of%20the%20object%20or%20environment.`.

[9] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[10] 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)*, IEEE, 2016, pp. 163–169.

[11] G. A. Borges and M.-J. Aldon, "Line extraction in 2d range images for mobile robotics," *Journal of intelligent and Robotic Systems*, vol. 40, pp. 267–297, 2004.

[12] A. Bryant and K. Cios, "Rnn-dbscan: A density-based clustering algorithm using reverse nearest neighbor density estimates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 6, pp. 1109–1121, 2017.

[13] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2013, pp. 160–172.

[14] CDC, *Global Road Safety — cdc.gov*, `https://www.cdc.gov/transportation-safety/global/index.html`, [Accessed 17-05-2025].

[15] P. Dahal, *Using Deep Neural Networks for Clustering — parasdahal.com*, Accessed 04-05-2025, 2019. [Online]. Available: `https://www.parasdahal.com/deep-clustering`.

[16] S. Dasgupta, "How fast is k-means?" In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*, Springer, 2003, pp. 735–735.

[17] M. Daszykowski, B. Walczak, and D. Massart, "Looking for natural patterns in data: Part 1. density-based approach," *Chemometrics and Intelligent laboratory systems*, vol. 56, no. 2, pp. 83–92, 2001.

[18] N. Donges, *4 Disadvantages of Neural Networks | Built In — builtin.com*, Accessed 04-05-2025, 2023. [Online]. Available: `https://builtin.com/data-science/disadvantages-neural-networks#:~:text=Neural%20networks%20usually%20require%20much,if%20you%20use%20other%20algorithms`.

[19] V. W. Eda Kavlakoglu, *What is k-means clustering? | IBM — ibm.com*, Accessed 04-05-2025, 2025. [Online]. Available: `https://www.ibm.com/think/topics/k-means-clustering#:~:text=Different%20clusters%20should%20be%20far,averaging%20values%20with%20a%20cluster`.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, 1996, pp. 226–231.

[21] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, 5th. Chichester, UK: John Wiley & Sons, 2011, Chapter 4: Hierarchical Clustering.

[22] F. Gao, C. Li, and B. Zhang, "A dynamic clustering algorithm for lidar obstacle detection of autonomous driving system," *IEEE Sensors Journal*, vol. 21, no. 22, pp. 25 922–25 930, 2021.

[23] A. Gunnargård and D. Odin, "Camera lidar active learning for object detecting deep neural networks," 2020.

[24] D. Guo, B. Qi, and C. Wang, "Fast clustering method of lidar point clouds from coarse-to-fine," *Infrared Physics & Technology*, vol. 129, p. 104 544, 2023.

[25] F. Hasecke, L. Hahn, and A. Kummert, "Flic: Fast lidar image clustering," *arXiv preprint arXiv:2003.00575*, 2020.

[26] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Chapter 14: Unsupervised Learning*, 2nd. New York: Springer, 2009, vol. 2.

[27] D. Held, D. Guillory, B. Rebsamen, S. Thrun, and S. Savarese, "A probabilistic framework for real-time 3d segmentation using spatial, temporal, and semantic cues.," in *Robotics: Science and Systems*, vol. 12, 2016.

[28] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[29] B. Kim, B. Choi, M. Yoo, H. Kim, and E. Kim, "Robust object segmentation using a multi-layer laser scanner," *Sensors*, vol. 14, no. 11, pp. 20 400–20 418, 2014.

[30] J. B. Kuipers, *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton university press, 1999.

[31] G. N. Lance and W. T. Williams, "A general theory of classificatory sorting strategies: 1. hierarchical systems," *The computer journal*, vol. 9, no. 4, pp. 373–380, 1967.

[32] Y. Li, C. Le Bihan, T. Pourtau, and T. Ristorcelli, "Insclustering: Instantly clustering lidar range measures for autonomous vehicle," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2020, pp. 1–6.

[33] L. Lin, Z. Haili, H. Zhen, M. Rongxin, L. Weikuang, and C. Zixuan, "Research on clustering of non-uniformly distributed point clouds in road scenes," in *Proceedings of the 2024 16th International Conference on Machine Learning and Computing*, 2024, pp. 376–381.

[34] Q. Lu, T. Tettamanti, D. Hörcher, and I. Varga, "The impact of autonomous vehicles on urban traffic network capacity: An experimental analysis by microscopic traffic simulation," *Transportation Letters*, vol. 12, no. 8, pp. 540–549, 2020.

[35] Luminar, *Luminar iris*, Accessed 03-05-2025, 2025. [Online]. Available: `https://www.luminartech.com/technology`.

[36] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3d object detection," in *2019 16th european radar conference (EuRAD)*, IEEE, 2019, pp. 129–132.

[37] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[38] T. Naous, S. Sarkar, A. Abid, and J. Zou, "Clustering plotted data by image segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 21 499–21 504.

[39] E. Nardelli, *Distributed k-d trees§*, 2003. [Online]. Available: `https://api.semanticscholar.org/CorpusID:15304762`.

[40] T. Nyström Johansson and O. Wellenstam, *Lidar clustering and shape extraction for automotive applications*, 2017.

[41] A. Pandharipande, C.-H. Cheng, J. Dauwels, *et al.*, "Sensing and machine learning for automotive perception: A review," *IEEE Sensors Journal*, vol. 23, no. 11, pp. 11 097–11 115, 2023.

[42] R. Pérez, F. Schubert, R. Rasshofer, and E. Biebl, "Deep learning radar object detection and classification for urban automotive scenarios," in *2019 Kleinheubach Conference*, IEEE, 2019, pp. 1–4.

[43] F. de Ponte Müller, "Survey on ranging sensors and cooperative techniques for relative positioning of vehicles," *Sensors*, vol. 17, no. 2, p. 271, 2017.

[44] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 658–666.

[45] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data mining and knowledge discovery*, vol. 2, pp. 169–194, 1998.

[46] SciKit Learn, *KMeans — scikit-learn.org*, Accessed 04-05-2025, 2025. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`.

[47] M. Taiebat, A. L. Brown, H. R. Safford, S. Qu, and M. Xu, "A review on energy, environmental, and sustainability implications of connected and automated vehicles," *Environmental science & technology*, vol. 52, no. 20, pp. 11 449–11 465, 2018.

[48] B. G. B. F. Waltz, "Intelligent machine vision," in Springer, 2001, ch. 16.

[49] J. Wang and X. Su, "An improved k-means clustering algorithm," in *2011 IEEE 3rd international conference on communication software and networks*, IEEE, 2011, pp. 44–46.

[50] X. Wang, H. Chen, and L. Wu, "Feature extraction of point clouds based on region clustering segmentation," *Multimedia tools and applications*, vol. 79, pp. 11 861–11 889, 2020.

[51] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[52] YellowScan, *LiDAR: What Is It and How Does It Work? | YellowScan — yellowscan.com*, Accessed 03-05-2025, 2025. [Online]. Available: `https://www.yellowscan.com/knowledge/how-does-lidar-work/`.

[53] G. Zamanakos, L. Tsochatzidis, A. Amanatiadis, and I. Pratikakis, "A comprehensive survey of lidar-based 3d object detection methods with deep learning for autonomous driving," *Computers & Graphics*, vol. 99, pp. 153–181, 2021.

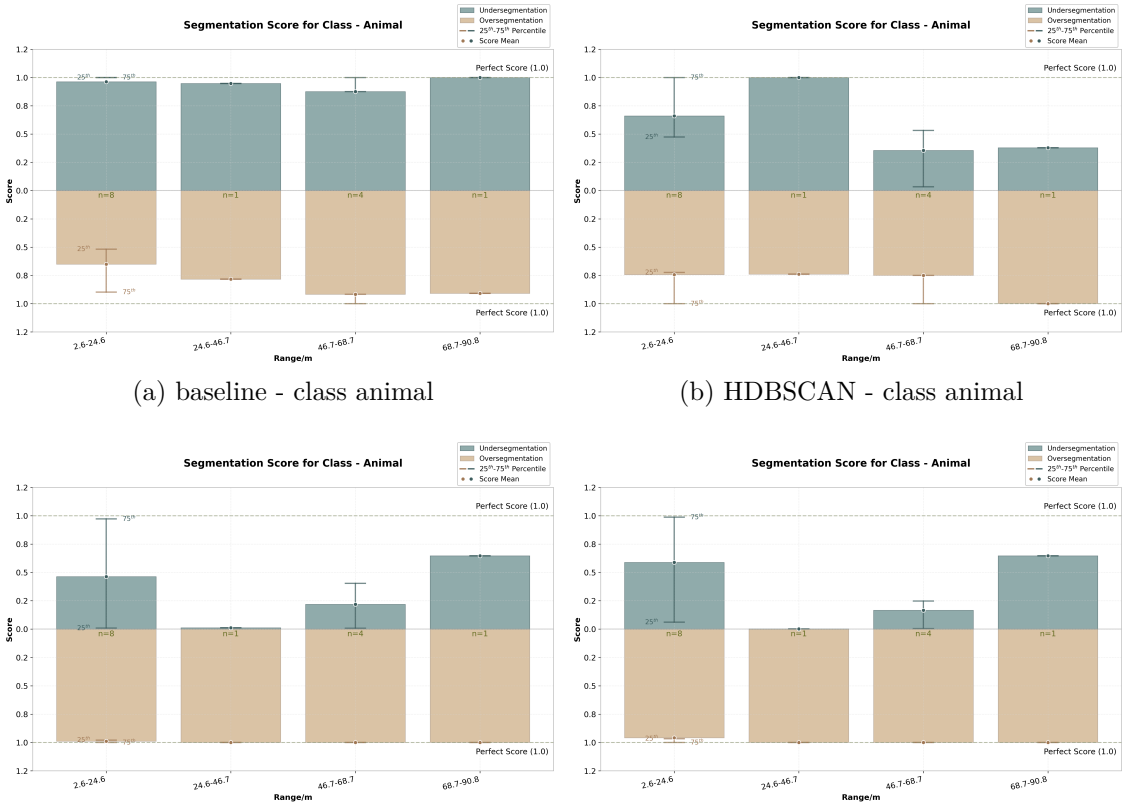
[54] X. Zhang and X. Huang, "Real-time fast channel clustering for lidar point cloud," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4103–4107, 2022.

[55] Y. Zhao, X. Zhang, and X. Huang, "A divide-and-merge point cloud clustering algorithm for lidar panoptic segmentation," in *2022 International conference on robotics and automation (ICRA)*, IEEE, 2022, pp. 7029–7035.

A

Appendix 1

A.1 Evaluation histograms for two more classes



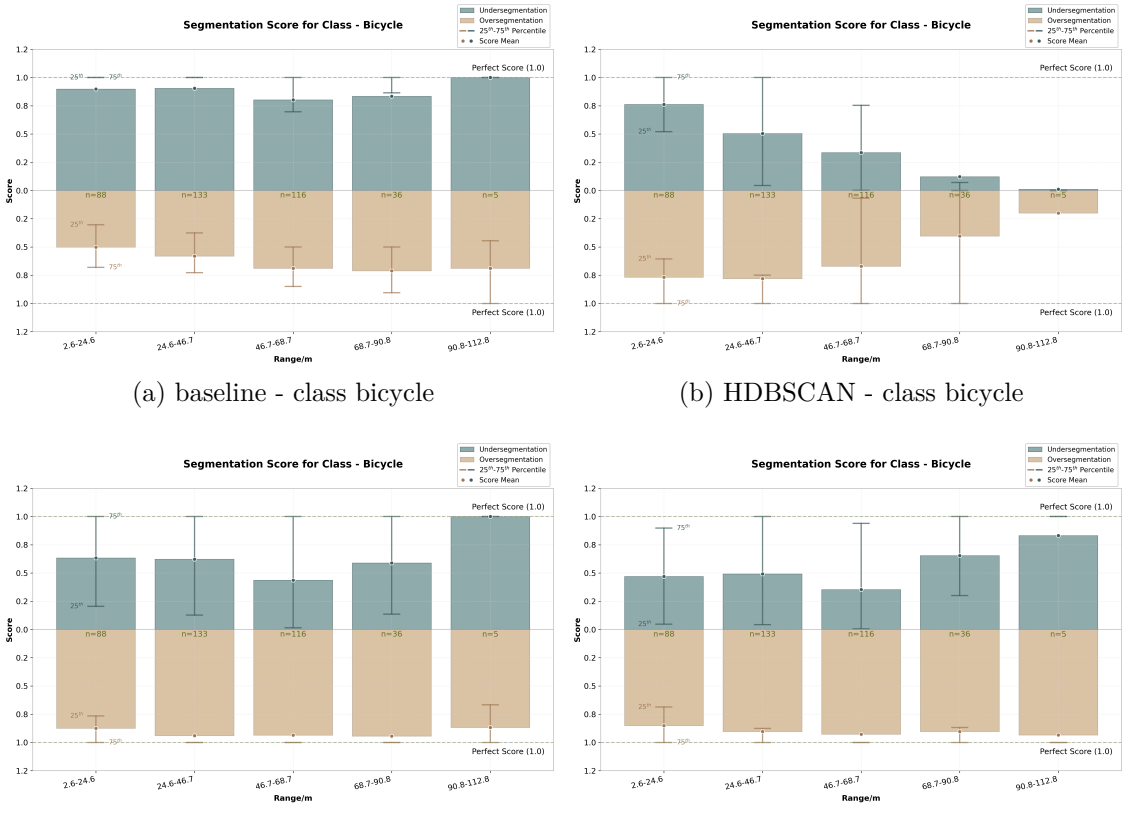
(a) baseline - class animal

(b) HDBSCAN - class animal

(c) RGAC - class animal

(d) HBAC - class animal

Figure A.1: Evaluation histograms for class animal.



(a) baseline - class bicycle

(b) HDBSCAN - class bicycle

(c) RGAC - class bicycle

(d) HBAC - class bicycle

Figure A.2: Evaluation histograms for class bicycle.