remote sensing

MDPI

Article

Real-Time Environmental Contour Construction Using 3D LiDAR and Image Recognition with Object Removal

Tzu-Jung Wu, Rong He and Chao-Chung Peng *

Department of Aeronautics and Astronautics, National Cheng Kung University, Tainan 70101, Taiwan; p46124324@gs.ncku.edu.tw (T.-J.W.); p48101021@gs.ncku.edu.tw (R.H.)
* Correspondence: ccpeng@mail.ncku.edu.tw; Tel.: +886-6-2757575 (ext. 63633)

Abstract: In recent years, due to the significant advancements in hardware sensors and software technologies, 3D environmental point cloud modeling has gradually been applied in the automation industry, autonomous vehicles, and construction engineering. With the high-precision measurements of 3D LiDAR, its point clouds can clearly reflect the geometric structure and features of the environment, thus enabling the creation of high-density 3D environmental point cloud models. However, due to the enormous quantity of high-density 3D point clouds, storing and processing these 3D data requires a considerable amount of memory and computing time. In light of this, this paper proposes a real-time 3D point cloud environmental contour modeling technique. The study uses the point cloud distribution from the 3D LiDAR body frame point cloud to establish structured edge features, thereby creating a 3D environmental contour point cloud map. Additionally, unstable objects such as vehicles will appear during the mapping process; these specific objects will be regarded as not part of the stable environmental model in this study. To address this issue, the study will further remove these objects from the 3D point cloud through image recognition and LiDAR heterogeneous matching, resulting in a higher quality 3D environmental contour point cloud map. This 3D environmental contour point cloud not only retains the recognizability of the environmental structure but also solves the problems of massive data storage and processing. Moreover, the method proposed in this study can achieve real-time realization without requiring the 3D point cloud to be organized in a structured order, making it applicable to unorganized 3D point cloud LiDAR sensors. Finally, the feasibility of the proposed method in practical applications is also verified through actual experimental data.

Keywords: 3D LiDAR; 3D point cloud processing; contour mapping; edge extraction; image object detection

check for updates

Citation: Wu, T.-J.; He, R.; Peng, C.-C. Real-Time Environmental Contour Construction Using 3D LiDAR and Image Recognition with Object Removal. *Remote Sens.* **2024**, *16*, 4513. https://doi.org/10.3390/rs16234513

Academic Editor: Pablo Rodríguez-Gonzálvez

Received: 29 September 2024
Revised: 23 November 2024
Accepted: 29 November 2024
Published: 1 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Background

Due to the rapid development of large-scale 3D environment modeling technology, environmental 3D modeling has been widely applied in various fields, including urban planning, autonomous vehicles, environmental monitoring, and military operations. With the fast expansion of urban areas and advancements in automation systems, the demand for real-time, detailed environmental information is increasing. Particularly, activities such as construction, renovation, and demolition lead to rapid changes in urban landscapes, highlighting the importance of fast mapping capabilities. Based on these demands, Light Detection and Ranging (LiDAR) sensors have become invaluable, as they can provide high-resolution spatial data at remarkable speeds, facilitating precise mapping of complex environments.

However, despite these advantages, high-density, large-scale point clouds generated by LiDAR often lead to enormous data volumes, imposing substantial memory and computational requirements for data processing, storage, and real-time applications. These issues become even more critical in resource-limited scenarios, such as edge computing

environments or portable platforms like autonomous vehicles and UAVs, where limited memory and computational power restrict efficient data processing and mapping capabilities. As the data stream grows, the need for fast, real-time feature extraction and recognition becomes paramount to ensure that only the most valuable information is retained for rapid decision-making.

Additionally, not all collected data contain essential structural information. In many cases, a 3D representation of the environment can be effectively achieved by focusing on key geometric characteristics, such as boundaries, edges, and corners, which capture the essential structure of the environment. By identifying and retaining only these critical features, it becomes possible to create a concise and meaningful model of the environment, reducing the amount of data needed without sacrificing accuracy.

Building on the challenges of large-scale 3D environment modeling and the advantages of feature extraction, this paper proposes a robust approach to efficiently capture and retain only essential structural information within 3D point clouds. The approach incorporates a series of techniques to filter and extract relevant environmental features, organized into three main parts: (1) vehicle detection through image recognition, (2) heterogeneous sensor object matching and removal, and (3) unorganized point cloud structural feature extraction. Through these processes, this work aims to build a real-time environmental 3D contour that includes the most valuable environmental characteristics and discards unreliable objects while requiring less memory storage.

1.2. Related Research

This paper focuses on a series of methods aimed at constructing a high-precision and rigid real-time mapping system for 3D environmental contours. The work includes key components such as vehicle detection, bounding box fitting, and environmental contour extraction. Vehicle detection is crucial for eliminating obstacles, bounding box fitting defines detected object space for accurate representation, and contour extraction facilitates the construction of a structured map. In recent years, advancements in each area have significantly contributed to environmental perception and real-time mapping accuracy. Below, the key methods for each component are summarized, highlighting their contributions to a cohesive mapping system.

1. Vehicle Detection

Most of the time, the vehicles that appear right now may disappear later. Under this scenario, the vehicles in the environment are treated as unreliable objects and should be removed during the mapping. With the rapid advancement of driverless technology, a wealth of research has been conducted on vehicle detection techniques, an essential capability for autonomous systems to accurately perceive and navigate their surroundings [1,2]. Vehicle detection in point clouds generally falls into three categories: LiDAR-based detection, camera-based detection, and LiDAR-Camera Fusion.

As point clouds captured by LiDAR are unorganized, LiDAR-based object detection relies heavily on clustering methods. This approach falls into two categories: point cloud classification and point cloud segmentation [2]. A common process involves detecting drivable areas, clustering the points, and classifying these clusters to identify vehicles [3,4]. Alternatively, some segmentation-based methods use convolutional neural networks (CNNs) to extract features from unstructured point clouds, enabling both obstacle identification and accurate vehicle localization [5,6].

Image data provides valuable color and texture details, as well as an organized pixel structure that facilitates object detection. In this field, Convolutional Neural Networks (CNNs) are widely used for classifying and recognizing vehicles, with approaches such as AlexNet [7], R-CNN [8], SSD (Single Shot MultiBox Detector) [9], and RetinaNet [10] being commonly applied. However, a major limitation of this method is the absence of depth information, which often makes depth-sensitive detection challenging.

LiDAR-Camera Fusion simultaneously leverages RGB information from images and depth information from LiDAR sensors. Therefore, this fusion is often chosen for object

detection and classification [11–14]. There are different categories of LiDAR-Camera Fusion: early fusion, late fusion, and deep learning-based fusion. In early fusion, data from the LiDAR and camera are fused at the raw data level. For example, object detection can be performed on images, and then the point cloud is projected onto the image to label and extract object points, which are subsequently re-projected back to the LiDAR frame This process ensures that the point cloud gains accurate object labels along with depth information. Xu et al. [15] enhance detection and real-time performance by combining YOLOv4-based image detection with point cloud clustering for path planning. Late fusion processes LiDAR and image data separately, then fuses the features at a later stage; Cho et al. [11] improved association methods by first processing the modalities independently and then using image classifications to refine object tracking.

Deep learning-based fusion methods use neural networks to learn the optimized way to combine LiDAR and camera data. Liu et al. [16] developed FuDNN, combining features from 2D images and 3D LiDAR data through a deep network to refine 3D box predictions. Similarly, Chen et al. [17] proposed Multi-View 3D networks (MV3D) that generate 3D object proposals from LiDAR's bird's eye view, projecting these onto multiple views for enhanced object classification and 3D box regression.

2. Bounding Box Fitting Algorithms

Due to the disorganized nature of point clouds, defining the spatial boundaries of detected objects is as crucial as the detection itself. Bounding boxes, which simplify complex object shapes using regular geometric representations, are commonly used for this purpose. L-shape fitting is a widely used method in bounding box algorithms, providing a straightforward approximation of object boundaries. However, its fixed shape constraints may not accommodate variations in vehicle shapes from multiple viewpoints, potentially impacting accuracy in real-world scenarios. As a result, complex algorithms are often combined with L-shape fitting to improve boundary accuracy.

The Oriented Bounding Box (OBB) is defined by an enclosing cuboid aligned with the object's main structure [18]. This approach is particularly suitable for cases where the detected objects are not aligned with any principal axis, such as in airborne point clouds [19,20]. On the other hand, the Minimum Volume Bounding Box (MVBB) algorithm focuses on tightly enclosing detected objects by minimizing the bounding box volume, which may use a range of bounding shapes, including the Minimum Volume Bounding Rectangle [21–23], Sphere [18], and Ellipsoid [24,25]. Selecting the appropriate geometric model based on the object's characteristics ensures an efficient and accurate boundary-fitting process.

3. Environmental Contour Extraction in Point Cloud

In the field of 3D mapping with LiDAR sensors, contour extraction has become an increasingly popular research trend, enabling applications like object recognition, classification, and environmental modeling [26–28]. Traditional methods often rely on the geometric or statistical properties of points to identify edges, such as curvature-based approaches that analyze local geometric properties of point clouds [29–31]. Pauly et al. [30] proposed an algorithm that estimates curvature and normal vectors by applying Principal Component Analysis (PCA), which maintains geometric characteristics while reducing data points, thus improving visualization and data processing efficiency [32].

Beyond geometry-based techniques, some researchers employ model-based approaches. Ramazan Alper et al. [33] combined RGB color data with RANSAC-based contour extraction to develop 3D urban construction models. Ji et al. [34] introduced a point cloud simplification method, Detail Feature Points Simplified Algorithm (DFPSA), to enhance accuracy while pursuing efficiency by evaluating the importance of various characteristic points and retaining details during point cloud simplification. Furthermore, Chen et al. [35] developed a point cloud feature regularization method to quickly extract boundary points by assessing neighborhood geometric features and smoothing boundary lines for a clear representation.

As deep learning has advanced, learning-based approaches have emerged to further improve contour extraction. Zang et.al. [36] proposed a novel learning-based network for point cloud contour extraction, known as LCE-Net, which efficiently extracts contours from large-scale 3D point clouds. This method exemplifies the potential of deep learning techniques in enhancing the accuracy and efficiency of contour extraction.

1.3. Main Work

This paper presents a comprehensive 3D environmental contour construction method, which consists of three primary steps: identifying and removing vehicles, which are regarded as temporary obstacles, to reduce uncertainties in the contour; extracting structural edges from the environment in the body frame for enhanced mapping clarity; and tracking vehicles in the accumulated frame to better define occupied spaces and ensure accurate removal in the global frame. Each step addresses specific challenges, contributing to the robustness and precision of 3D mapping for autonomous driving and environmental monitoring.

The process begins by aligning 2D images with 3D LiDAR point clouds for precise vehicle detection. A pre-trained YOLO model [37], a deep learning-based object detection model known for its real-time performance, is applied to 2D images to detect vehicles. These detections are then mapped onto the corresponding 3D LiDAR point clouds. This fusion overcomes the limitations of single-sensor data (e.g., camera or LiDAR alone) in dynamic object detection and localization by combining YOLO-based image detection with point cloud clustering, balancing accuracy and computational efficiency over purely deep learning-based fusion. Identifying vehicles enables their removal from the scene to prevent interference in contour construction, enhancing map quality by isolating temporary objects from static environmental features.

Next, this work presents an Adaptive DBSCAN Algorithm that clusters and detects vehicles at varying distances within the 3D LiDAR point clouds. To establish an appropriate parameter tuning strategy, this paper proposes a strategy that clarifies how point density changes with distance, resulting in a function to guide parameter adjustment. Addressing challenges with the fixed clustering parameters in traditional DBSCAN, this algorithm dynamically adjusts its parameters based on distance, improving vehicle identification accuracy across different spatial contexts. Once vehicles are identified, they are removed from the body frame, enabling immediate structural edge extraction using Principal Component Analysis (PCA). By removing temporary obstacles early, PCA focuses on emphasizing structural edges within the body frame, providing a clear and efficient local representation of the environment.

Simultaneously, vehicle points detected in each frame are collected during the mapping process to enable consistent tracking across multiple LiDAR frames in the global frame. The occupied regions of these tracked points are then modeled as Minimum Volume Enclosing Ellipsoids (MVEE). This tracking process is essential to address discrepancies in the fields of view (FOV) between the camera and LiDAR. Without tracking, obstacles removed in the body frame due to temporary mismatches in sensor coverage may reappear during global map construction. By accumulating observations across frames, this approach ensures consistent removal of dynamic objects in the global frame, preventing such artifacts.

The use of MVEE for modeling tracked objects further enhances the clarity and stability of the global map by overcoming the limitations of rigid, L-shaped constraints. This allows for precise separation of static structures while ensuring that contours remain free from interference by temporary obstacles.

Unlike traditional 3D contour extraction methods that may overlook the modeling uncertainties caused by temporary obstacles like vehicles. This work tends to minimize these uncertainties while retaining essential environmental features. These structural data are critical for autonomous navigation as they highlight significant features within the environment with less memory required.

The contributions of this work include (a) an effective LiDAR-camera fusion method for robust vehicle identification and localization, (b) a parameter-adaptive DBSCAN for

accurate object clustering and removing in body frame, (c) an MVEE-based approach for consistent tracking and removal of dynamic objects across accumulated frames, and (d) a contour extraction technique that emphasizes environmental features while minimizing interference from temporary obstacles. This integrated solution enhances map accuracy and stability, which is crucial for autonomous driving and navigation applications.

The algorithm was tested on image and LiDAR data from the popular KITTI dataset, and experimental results demonstrated its effectiveness in generating precise 3D environmental contours. The flow chart of the proposed process is shown in Figure 1.

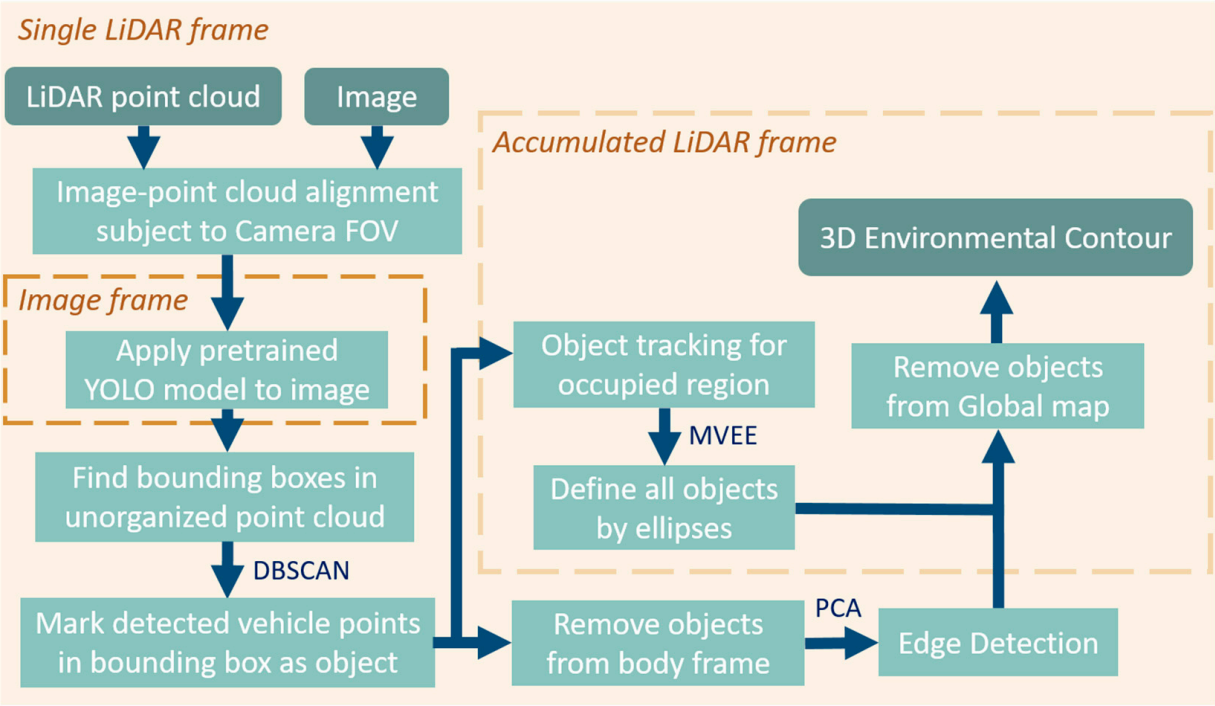


Figure 1. The flow diagram of the proposed process. This workflow results in a 3D environmental contour generated from a single LiDAR frame.

2. Methods

2.1. LiDAR-Camera Fusion

LiDAR-camera fusion is an efficient way to combine the RGB information of the camera data and the depth information of the LiDAR point cloud data. Based on the informative color image, neural network-related technologies are capable of providing great accuracy and efficiency for object detection. Rather than directly taking object detection based on the 3D LiDAR point clouds, in this paper, the result of object detection based on the pre-trained YOLOv8 model was integrated together with the 3D point clouds; it can reach a preliminary object spatial extraction in real time.

To achieve the object spatial extraction, heterogeneous sensor fusion is needed. The data of the image and point cloud in the KITTI dataset lack correspondence, to utilize the detected bounding boxes in the image frame to the point cloud, projecting the point cloud onto the image plane will be a compulsory step. The coordinate transformation for these two heterogeneous sensor alignments is calculated by the following equation:

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \, {}^{Camera}_{LiDAR}\mathbf{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (1)$$

where **K** is the intrinsic matrix of the camera, ${}^{Camera}_{LiDAR}\mathbf{T}$ is a transformation matrix from LiDAR coordinate to camera coordinate, Z_c is the depth information of the camera. This transformation relation maps the 3D point cloud coordinate onto the 2D image plane. Thus, the bounding box region in the image plane can be recognized in the point cloud coordinate, as shown in Figure 2. Refer to the heterogeneous sensor alignment as summarized in Figure 2, Figure 2a shows the image object detection using YOLOv8; Figure 2b illustrates the 3D LiDAR point cloud projection onto the 2D image via (2), and Figure 2c expresses the final alignment result based on Figure 2b,c, which demonstrates the preliminary object spatial extraction procedure.

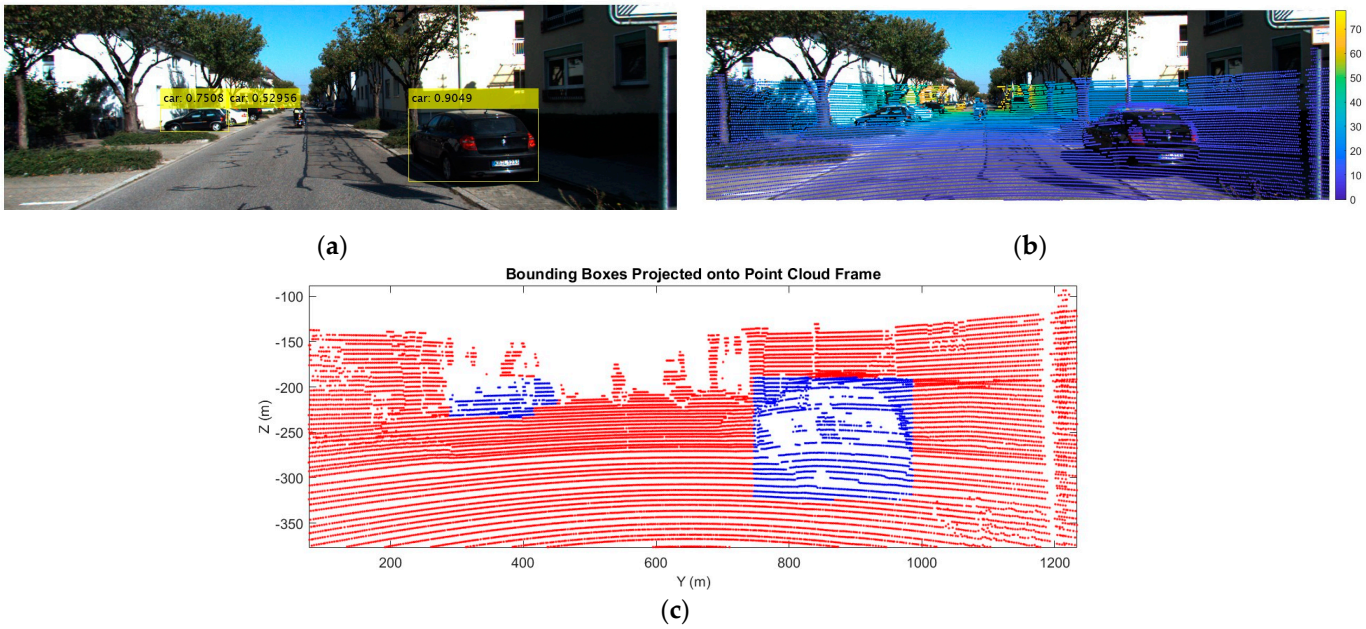


Figure 2. The process of mapping the bounding box to the point cloud. (**a**) The image with car detection bounding boxes by YOLO v8; (**b**) The projection of point cloud on the image plane; (**c**) The result of defining the region of the bounding boxes in 3D point cloud. The blue points indicate the points projected on by the bounding boxes and the red points represent the point cloud in camera's FOV but not within the bounding boxes detected by YOLO.

2.2. DBSCAN Algorithm for Object Points Extraction Refinement

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an algorithm that clusters a set of data points into different groups based on their spatial density [38]. Note that the bounding box regions, projected onto the 3D point cloud, encompass not only the vehicles but also background areas, as depicted in Figure 3. To separate the vehicle points from the background points, the popular DBSCAN algorithm is applied to identify the main clusters within the set of bounding box points, thereby defining the main object points.

The algorithm requires two parameters: *Eps* and *MinPts*. It calculates the *Eps* distance for each point in the dataset. If the number of points within this *Eps* distance exceeds the *MinPts* threshold, all these points are considered part of the same cluster. Figure 4 demonstrates the basic algorithm, where *Eps* is set to ε and *MinPts* is 4. In this example, point C falls within the Eps range of the B cluster. Although the number of points in the C cluster is fewer than 4, they are still considered part of the B cluster. Consequently, the B and C clusters merge into a single cluster, represented by orange points in the figure. Conversely, no points from the A cluster fall within the orange region, so the A cluster, represented by blue points, is defined as a separate cluster. Point D, which does not belong to any cluster, is identified as an outlier point.

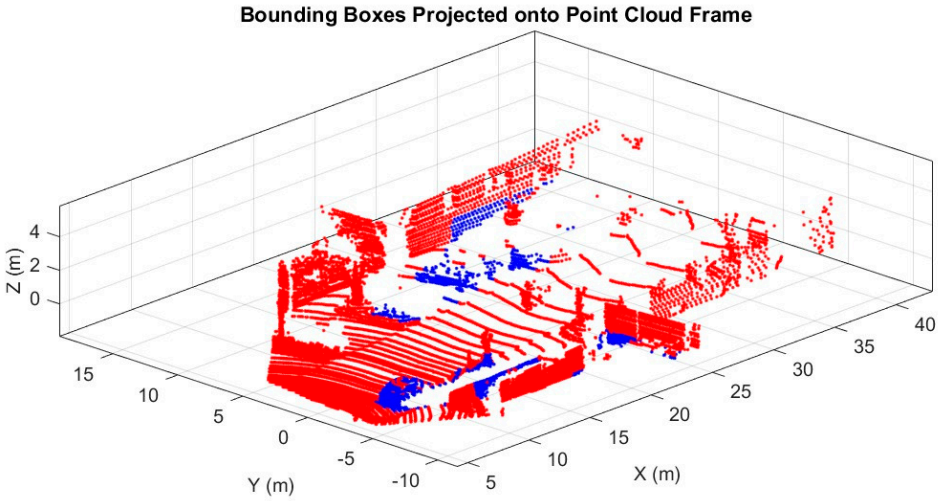


Figure 3. The result of defining the region of the bounding boxes in 3D point cloud. The blue points indicate the points projected on by the bounding boxes and the red points represent the point cloud in camera's FOV but not within the bounding boxes.

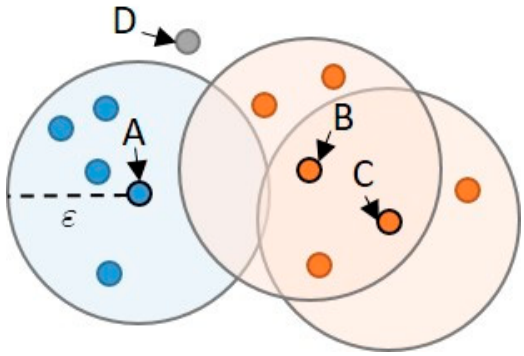
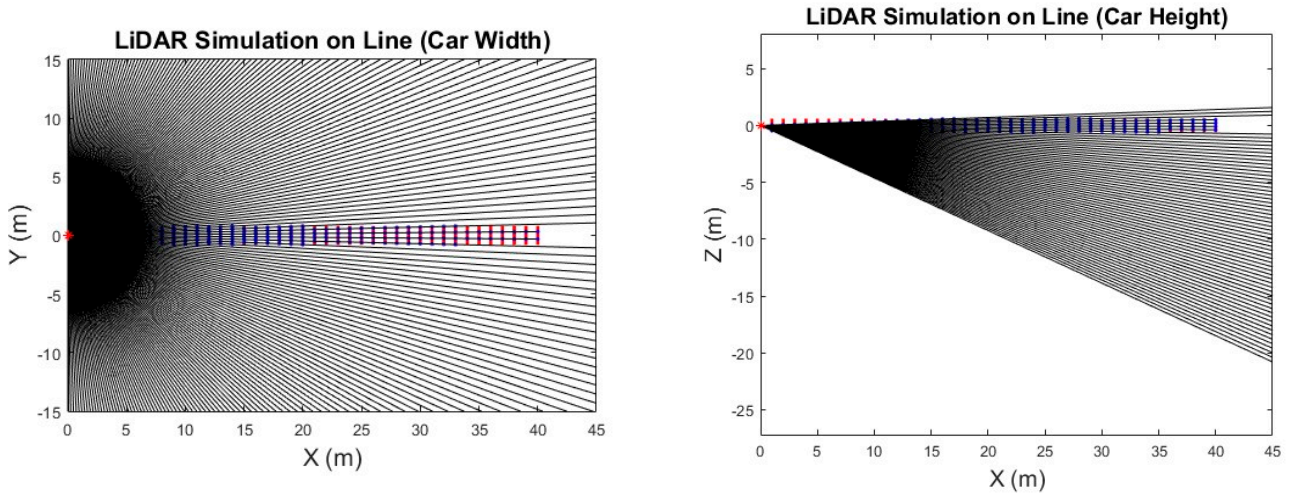


Figure 4. Schematic of DBSCAN algorithm.

To address the varying distribution of points on vehicles at different distances from the LiDAR sensor, this work presents an algorithm that adjusts the *MinPts* parameter in the DBSCAN algorithm based on the distance. As vehicles get farther away, the point distributions become sparser, while closer vehicles have denser point distributions. This necessitates different DBSCAN parameters for different distances.

In the following, a scenario is considered to determine the number of points detected on vehicles. The geometry of vehicles considered both the side and back faces, with the back face being smaller. Point clusters exceeding the expected count for the back face were used to identify vehicles. The simulation incorporated the specifications of the LiDAR sensor, specifically the Velodyne HDL-64E used in the KITTI dataset. This sensor has a horizontal angular resolution of 0.08 degrees across a 360-degree field of view (FOV) and a vertical FOV of 26.8 degrees (from 2 to −24.8 degrees), subdivided into 64 lines. Standard vehicle dimensions of 1.6 m in width and 1.2 m in height were assumed. Lines with these dimensions were placed at various distances from the LiDAR sensor to calculate the number of points detected at each distance.

The simulation scenarios are illustrated in Figures 5a and 5b, respectively. This setup demonstrates how the number of detected points varies with distance from the LiDAR sensor.



(**a**) (**b**)

Figure 5. The demonstration of the number of detected points varies with distance. (**a**) Simulation on car width; (**b**) Simulation on car height. The red star represents the LiDAR sensor's position, red lines denote the width and height lines at different distances, respectively, and the blue points along the lines mark the detected points.

After determining the number of points at different distances, nonlinear curve fitting is applied to model the relationship between the number of points and distance. For the distribution shape of the data points varies with distance, the exponential function is chosen as the model function (2):

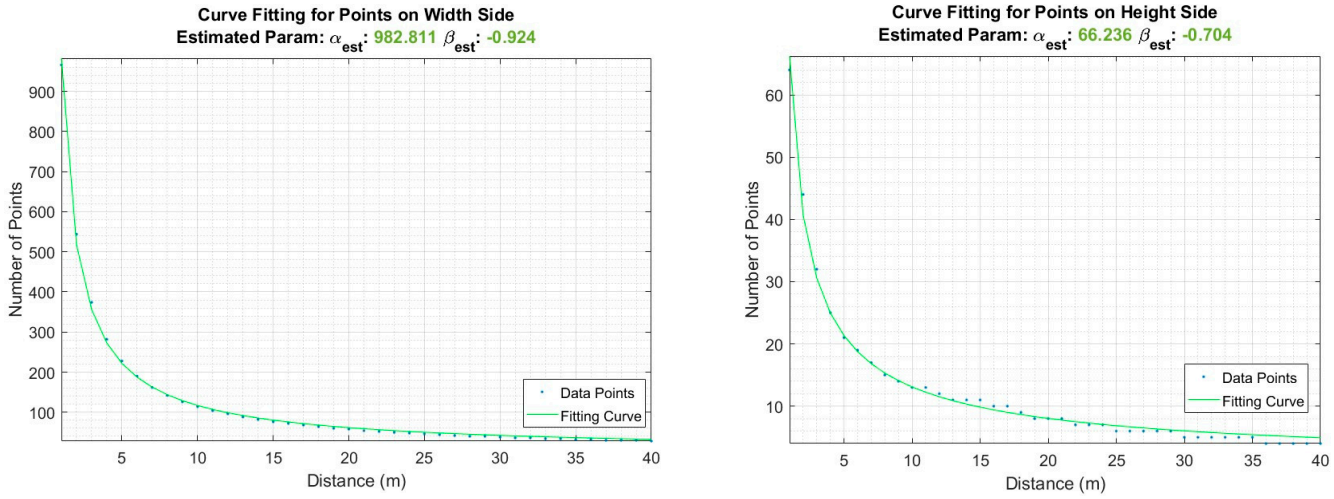
$$y = \alpha x^{\beta} \tag{2}$$

where y represents the number of points, and x represents the distance from the LiDAR sensor.

After the optimization of the parameters of the model function (As shown in Figure 6), the result of the fitted models for width and height can be represented as (3) and (4), respectively.

$$y = 982.811x^{-0.924} \tag{3}$$

$$y = 66.236x^{-0.704} \tag{4}$$



(**a**) (**b**)

Figure 6. The result of the curve fitting. (**a**) The curve fitting parameter of the points on the width side; (**b**) The curve fitting parameter of the points on the height side.

For the plane composed of height and width, the total number of points is:

$$points_{backplane} = points_{width} \times points_{height} \quad (5)$$

Then, the number of points at different distances can be found, and the relationship between the backplane and the distance is shown in Figure 7.

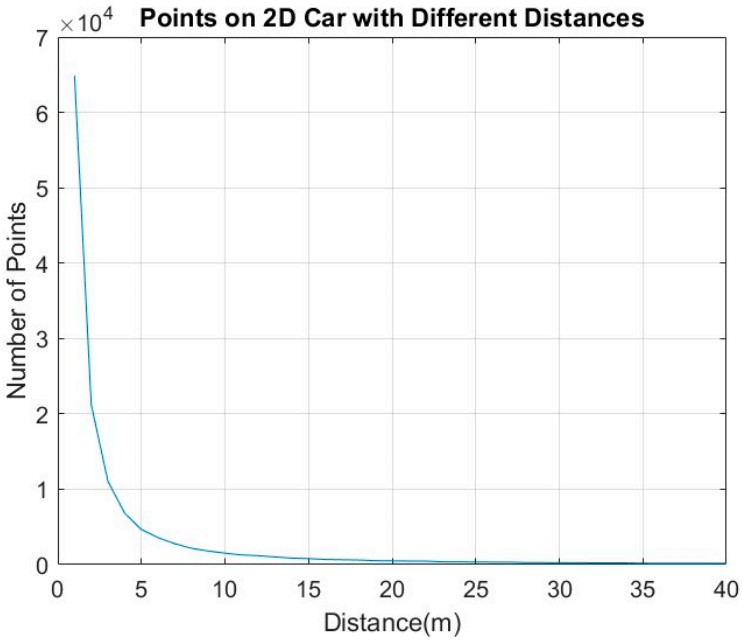


Figure 7. Number of points detected on 2D car rear with different distances.

The previous simulation results of LiDAR scanning the rear of the vehicle were based on the assumption that the rear is rectangular. However, the vehicle has an irregular geometric shape and transparent surfaces, such as an unusable area, as demonstrated in Figure 8.

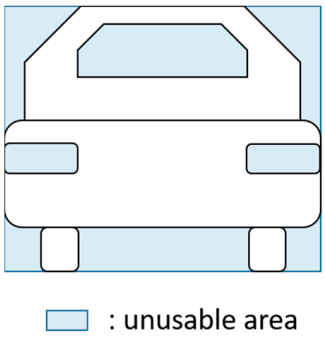


Figure 8. The Schematic of the ratio of the unusable area.

To more accurately reflect the real situation, a correction factor (CF) is applied to the simulated results to account for the unusable areas. In this case, the correction factor is set to 0.5. Then, the number of points is corrected as follows:

$$CarPoint = CF \times RectangularPoint \quad (6)$$

Based on this simulation, the parameter *MinPts* is set to a value that reliably identifies a vehicle cluster within the bounding box. Since the algorithm's logic requires each point to have a sufficient number of neighbors within a specific distance to form a cluster, fixing the *Eps* parameter to the vehicle's width (1.6 m) ensures that all relevant points within the vehicle's physical boundaries are included in the cluster.

However, smaller bounding boxes, often projected by fewer LiDAR points within the camera's FOV, may lead to an insufficient number of points for clustering. As a result, despite the Adaptive DBSCAN algorithm's adaptability to varying distances, it can become inefficient due to its sensitivity to the point density within the bounding box. To address this limitation, the algorithm employs a fallback mechanism: a secondary DBSCAN run with looser parameters is performed. This ensures that at least one cluster is found within each bounding box, and the cluster with the highest number of points is chosen to identify the detected vehicle.

The pseudo-code of the algorithm is illustrated as follows (Algorithm 1):

Algorithm 1 Adaptive DBSCAN-based Vehicle Point Extraction Algorithm

Input:
BoundingBoxPoint: Points enclosed by each bounding box within camera's FOV.
$[\alpha_1, \beta_1, \alpha_2, \beta_2]$: Parameters for calculating the number of points on the vehicle's width and height based on distance.

Output:
*VehiclePoint***:** Points classified as belonging to vehicles.
BoundingBoxBoundary: Refined bounding boxes defined by detected vehicle.

01: **For** each bounding box *i* from 1 to *NumberofBboxes*:
02: Find points enclosed by bounding box *i* as *BoundingBoxPoint*
02: Calculate *distance* from the sensor to the mean position of *BoundingBoxPoint*
03: Determine the expected number of points *Pts* based on *distance*:

$$PtsWidth = \alpha_1\ distance^{\beta_1}$$
$$PtsHeight = \alpha_2\ distance^{\beta_2}$$
$$Pts = PtsWidth \times PtsHeight$$

04: Set the DBSCAN parameters: $Eps = 1.6; MinPts = round(CF \times Pts)$
05: Apply Adaptive DBSCAN to cluster *BoundingBoxPoint*

06: **If** valid cluster *j* are detected:
07: Identify points belonging to the cluster *j* as *VehiclePoint*
08: **Else:**
09: Apply DBSCAN with looser DBSCAN parameters to *BoundingBoxPoint*.
10: Set cluster *j* with the highest number of points.
11: Identify points belonging to the cluster *j* as *VehiclePoint*.
12: **End If**

13: Calculate *BoundingBoxBoundary* enclosing *VehiclePoint*

14: **End For**

15: **Return** *BoundingBoxBoundary*, *VehiclePoint*

Finally, the clusters within the bounding boxes are identified as the vehicles in the frame, depicted as colored points in Figure 9. The corresponding vehicles detected by image recognition are shown as well, by the comparison, one can clearly see that the vehicle points detected by the Adaptive DBSCAN algorithm are accurate.

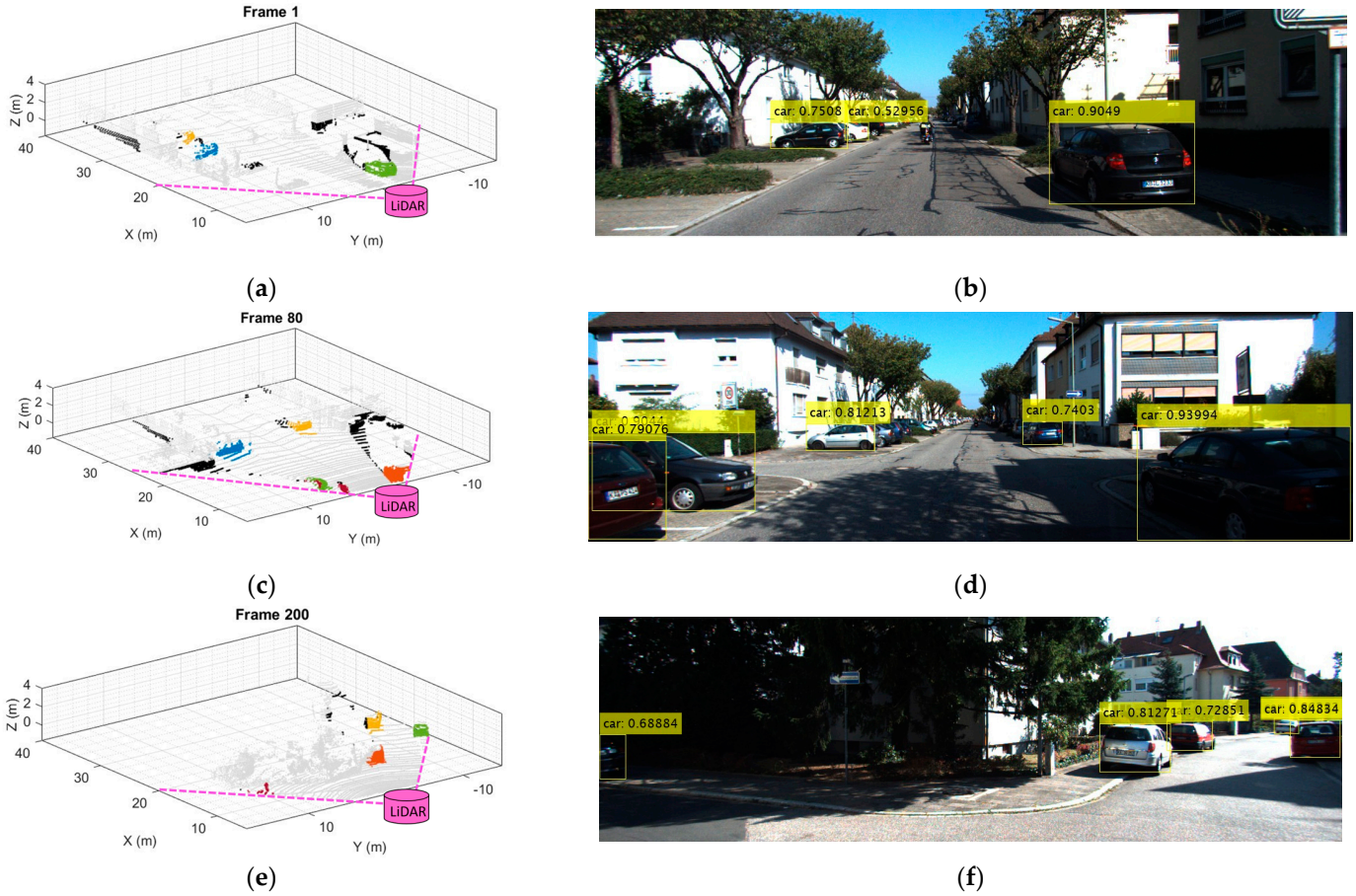
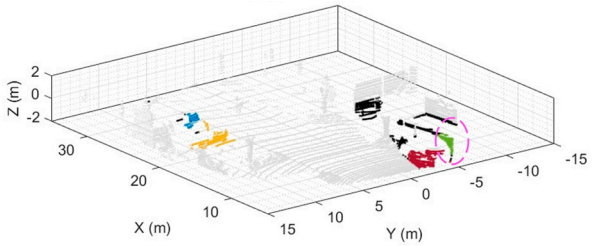
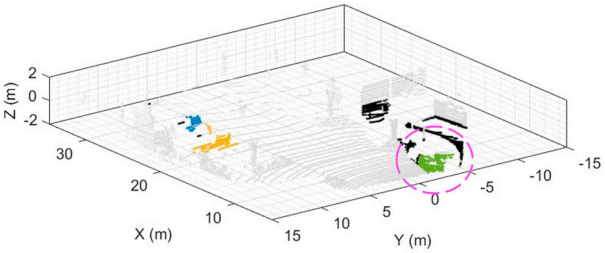
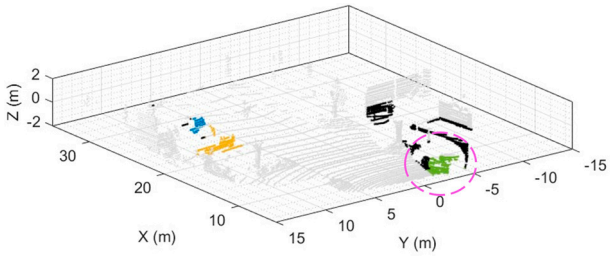
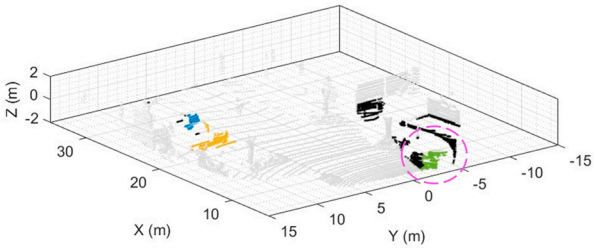
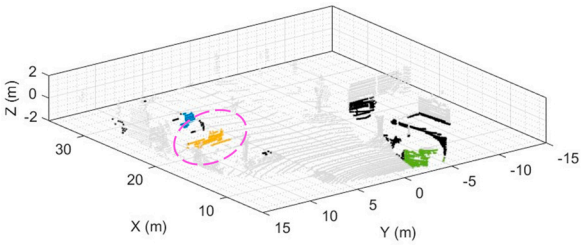


Figure 9. The results of the Adaptive DBSCAN algorithm for vehicle detection compared to the image recognition results. Grey points represent areas outside the detected bounding boxes, while black points indicate those within the bounding boxes but are classified as outliers by the DBSCAN algorithm. Color points highlight the different vehicles successfully identified by the algorithm. The results are displayed for Frame 1 in (**a**) point cloud and (**b**) color image, Frame 80 in (**c**) point cloud and (**d**) color image, and Frame 200 in (**e**) point cloud and (**f**) color image, respectively.

To demonstrate the effectiveness of the Adaptive DBSCAN Algorithm, this work compares the results with and without its use in frame 5, as shown in Table 1. Vehicle detection accuracy can be highly sensitive to the choice of the DBSCAN parameter *MinPts*, which determines the minimum number of points required to form a cluster. When *MinPts* is chosen empirically, setting it too low (e.g., *MinPts* = 300) makes clustering overly inclusive, as clusters can form with fewer points, increasing the likelihood that background points will be mistakenly grouped as vehicles, resulting in false detections. On the contrary, as *MinPts* increases, accuracy improves at various levels, as the point count for vehicles in this scenario is close to this range, with 400 yielding the best detection performance. However, a fixed *MinPts* remains problematic since varying distances and vehicle geometries affect point density, requiring different values to maintain accuracy across scenarios.

Table 1. Comparison of detection results with and without Adaptive DBSCAN Algorithm.

MinPts	Detected Result	
300		Bounding Boxes: 3 Detected Vehicles: 4
400		Bounding Boxes: 3 Detected Vehicles: 3
500		Bounding Boxes: 3 Detected Vehicles: 3
600		Bounding Boxes: 3 Detected Vehicles: 3
Proposed Adaptive DBSCAN Algorithm		Bounding Boxes: 3 Detected Vehicles: 3

With the Adaptive DBSCAN algorithm, the proposed scheme dynamically adjusts the DBSCAN parameters based on distance, allowing for accurate detection even for distant vehicles (presented in yellow), which remain distinguishable from background points (presented in black). In contrast, without the adaptive algorithm, detection relies on a single, fixed-parameter setting. This limitation often necessitates the secondary fallback mechanism—a secondary DBSCAN run with looser parameters to capture distant vehicles. While this fallback ensures at least one vehicle is detected per bounding box, it performs poorly with distant and sparse point clouds, where fixed parameters often fail to cluster points correctly due to their sparse distribution.

This comparison highlights the critical role of the Adaptive DBSCAN Algorithm in fine-tuning the detection process, enhancing accuracy by adjusting for distance, and ensuring reliable identification of vehicles in varied spatial conditions.

2.3. Object Bounding Box Tracking Behind the Camera FOV

After introducing the DBSCAN clustering method, one challenge the algorithm faces when recognizing objects from image data is the difference in the field of view (FOV) between the camera and the LiDAR. For example, the camera only captures the back side of the car. However, the LiDAR might capture the front side of the car.

To address this, the algorithm must not only identify and remove points corresponding to objects detected within the camera's FOV but also track these objects across multiple frames to define the regions they are going to occupy. This ensures the removal of the object points from the LiDAR's FOV. Figure 10 illustrates the impact of the difference between the LiDAR's FOV and the camera's FOV on object removal. In part (a), the blue points represent the vehicle points detected by image recognition within the camera's FOV, which are subsequently removed. However, in part (b), as the sensors move, the orange points—representing the vehicle—are detected and mapped again by the LiDAR despite having been removed in the previous step.

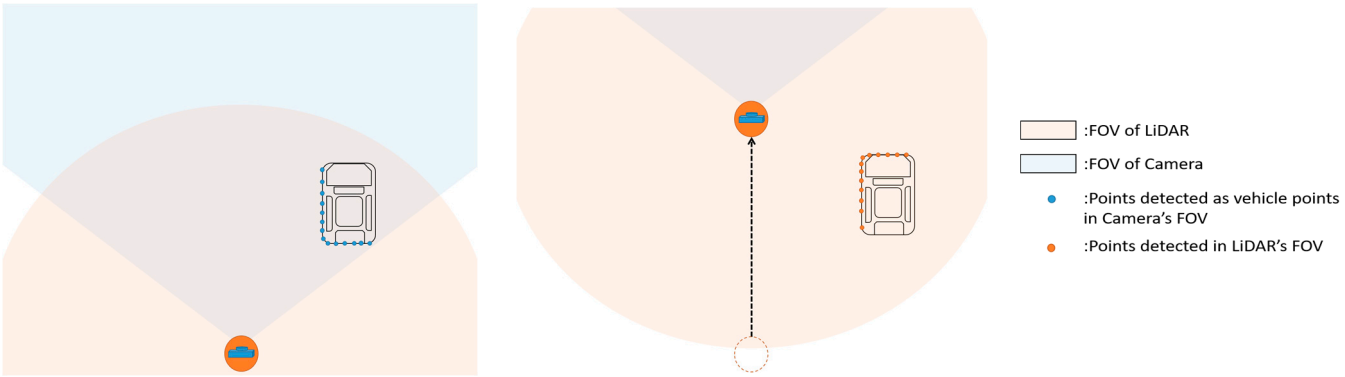


Figure 10. The demonstration of the impact of the difference between the LiDAR's FOV and the camera's FOV on object removal. (**a**) Points removed within the camera's FOV; (**b**) Points mapped within the LiDAR's FOV.

According to Section 2.1, object points are initially defined, allowing us to derive the boundary coordinates that form the bounding box around each car. The following compares the mapping process without and with object bounding box tracking. As shown in Figure 11, parts (a) and (b) represent different views of the mapping results without object bounding box tracking. The red points indicate the route of the vehicle sensing location. As the sensor vehicle moves from right to left, the vehicle points are removed as the bounding boxes are constructed. However, when the sensor vehicle moves further left, the bounding boxes are no longer created because the vehicles are outside the camera's FOV. In contrast, when bounding boxes are continuously tracked, all points within these bounding boxes—recognized as the target vehicles, which should be consistently removed. This feature is illustrated in parts (c) and (d) from different views. This experiment proved the necessity of the mapping process with an object-bound box tracking mechanism behind the camera's FOV. Without the aid of the proposed tracking mechanism, those should be removed vehicles will be regenerated by 3D LiDAR, which is not the preferred expected result.

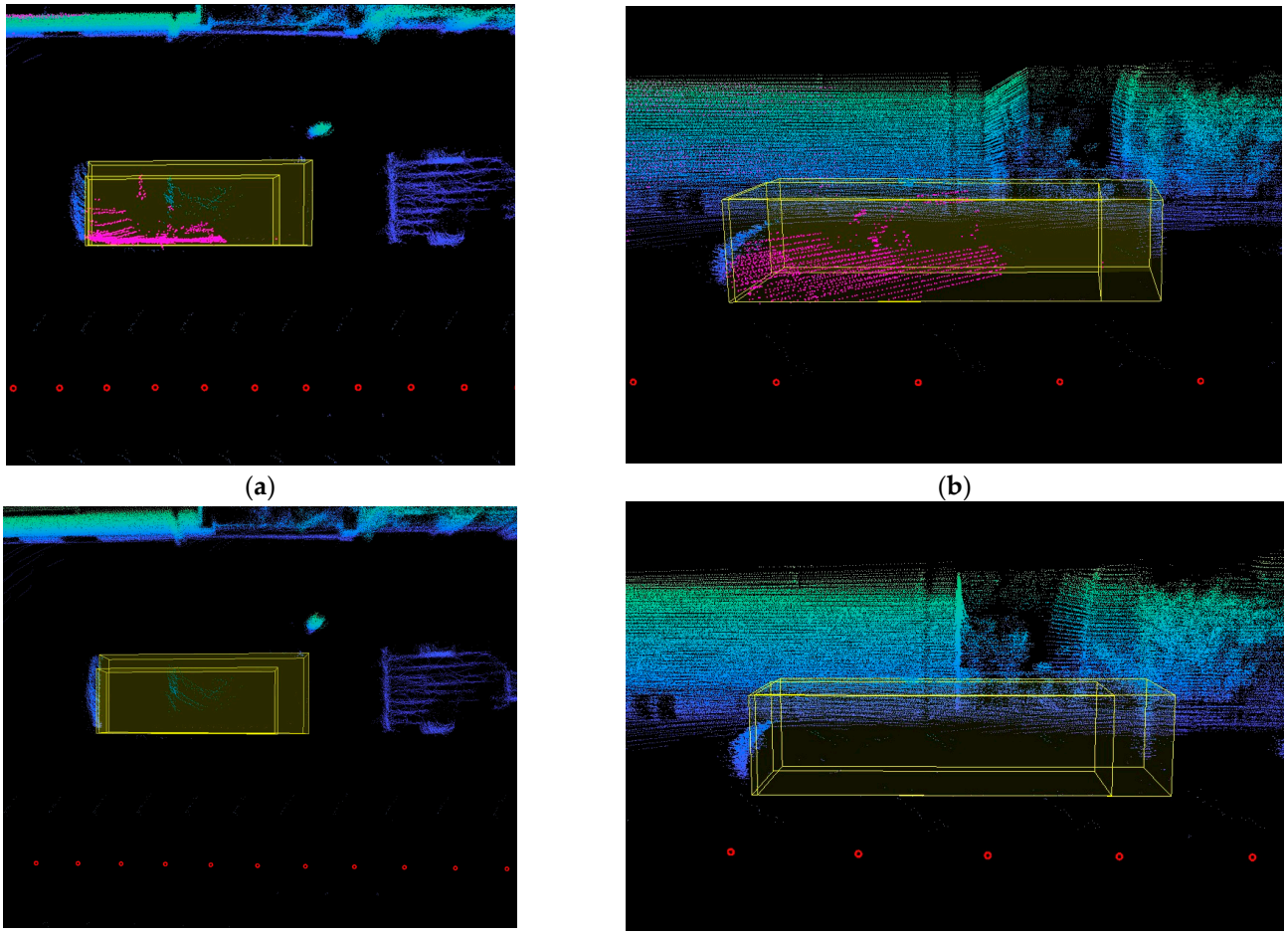


Figure 11. The demonstration of the mapping process without and with object bounding box tracking. (**a**,**b**) Mapping result of the process without object bounding box tracking; (**c**,**d**) Mapping result of process with object bounding box tracking. The pink points indicate points within the bounding boxes that are not eliminated in the first stage due to the difference in the FOV of the sensors.

To accurately remove all vehicle points, an approach is adopted that tracks all points within the bounding boxes during the map-building process. Global bounding boxes are then established, and all points within these boxes are removed in a single step.

2.4. Object Enclosed Ellipsoid

Compared to the traditional rectangular bounding box, which often fails to accurately capture the complex shape of a car, this study employs the MVEE method [24] to more precisely determine the area occupied by the vehicle. The MVEE method identifies an ellipsoid that encloses a set of data points with the smallest possible volume, offering a mathematical representation that makes it easier to identify points within or outside the defined region. This characteristic provides a significant advantage for object removal tasks, as points inside the ellipsoid can be reliably excluded. As a result, this method enables a more accurate elimination of non-static elements, improving the overall clarity and efficiency of environmental modeling.

The MVEE problem is formulated as follows:

given a set of 3-dimensional set of data points $\mathbf{Q}$:

$$\mathbf{Q} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \tag{7}$$

an ellipsoid E in center form can be described by

$$E = \left\{ \mathbf{x} \in \mathbb{R}^n \middle| (\mathbf{x} - \mathbf{c})^T \mathbf{A} (\mathbf{x} - \mathbf{c}) \leq 1 \right\} \tag{8}$$

where $\mathbf{A} \in S_n^{++}$ is a positive-definite matrix and $\mathbf{c} \in \mathbb{R}^n$ is the center of ellipsoid.

The volume of the ellipsoid has the following relation:

$$Vol(E) = v_0 \det\left(\mathbf{A}^{-1}\right)^{\frac{1}{2}} \tag{9}$$

where v_0 is the volume of the unit hypersphere in n dimensions. Thus, the optimization problem becomes:

$$\min_{A,c} \det\left(\mathbf{A}^{-1}\right) \tag{10}$$

$$\text{subject to } (\mathbf{x}_i - \mathbf{c})^T \mathbf{A} (\mathbf{x}_i - \mathbf{c}) \leq 1 \ , \ \forall i = 1, \ldots, n \tag{11}$$

To simplify the problem, the above demand can be transformed into a convex optimization problem using the dual formulation. The dual problem involves maximizing a function of a new variable u, which represents the weights of the data points in defining the ellipsoid. The MVEE problem solved by the Khachiyan Algorithm is described in pseudo-code illustrated as follows (Algorithm 2):

Algorithm 2 MVEE algorithm

Input: **Q** (Data points matrix of size $(d+1) \times n$), tolerance
Output: **A** ($d \times d$ matrix for the ellipsoid), **c** (center of the ellipsoid)

01: Initialize **u** as a vector of size n with each element set to $1/n$

02: **While** *err* > *tolerance* **do**
03: Compute **X** = **Q** * diag(**u**) * **Q'**
04: Compute **M** = diag(**Q'*** inv(**X**) * **Q**)
05: Find maximum value *maximum* in **M**, and its index *j*
06: Calculate *step_size* = (*maximum* − *d* − 1)/((*d* + 1) * (*maximum* − 1))
07: Update **new_u** by scaling **u** by (1 − *step_size*)
08: Set **new_u** = **new_u** + *step_size*
09: Compute *err* = norm(**new_u** − **u**)
10: Update **u** to **new_u**

11: **Compute Ellipsoid Parameters:**
12: Set **U** = diag(**u**)
13: Calculate A = (1/*d*) * inv(**P** * **U** * **P'** − (**P** * **u**) * (**P** * **u**)')
14: Set **c** = **P** * **u**

15: **Return A**, **c**

However, due to the L-shape distribution of the data points, using vehicle points to define an MVEE may not fully enclose the entire car. This issue arises because the L-shape can cause the MVEE to be skewed or miss parts of the vehicle when relying solely on scattered object points. To address this issue, only the eight corners of the bounding box are used to build the MVEE. This approach ensures that the MVEE fully encloses the car by leveraging the bounding box's structure to evenly distribute points within the vehicle's occupied space. Additionally, using only the eight corner points is computationally more efficient than incorporating all the object points. This method reduces the complexity of the MVEE calculation while maintaining accuracy, effectively mitigating the issues posed by the L-shape distribution. Furthermore, a merged bounding box strategy is adopted by overlapping bounding boxes for the same object across consecutive frames. This strategy merges the bounding boxes under the following conditions:

$$\begin{aligned} x_{1,max} &\geq x_{2,min} \; and \; x_{1,min} \leq x_{2,max} \\ y_{1,max} &\geq y_{2,min} \; and \; y_{1,min} \leq y_{2,max} \\ z_{1,max} &\geq z_{2,min} \; and \; z_{1,min} \leq z_{2,max} \end{aligned} \tag{12}$$

If these conditions are met simultaneously, the bounding boxes are considered to be intersecting, and then a merged bounding box, by taking the outermost coordinates from both bounding boxes, will be created. This process, for example, is expressed mathematically as follows:

$$\begin{aligned} x_{merged,\min} &= min(x_{1,\min}, x_{2,min}), x_{merged,max} = max(x_{1,max}, x_{2,max}) \\ y_{merged,min} &= min(y_{1,min}, y_{2,min}), \; y_{merged,max} = max(y_{1,max}, y_{2,max}) \\ z_{merged,min} &= min(z_{1,min}, z_{2,min}) \; , \; z_{merged,max} = max(z_{1,max}, z_{2,max}) \end{aligned} \tag{13}$$

The schematic of the merging bounding box strategy is shown in Figure 12. Using the merged bounding box ensures a more accurate representation of the space occupied by the car.

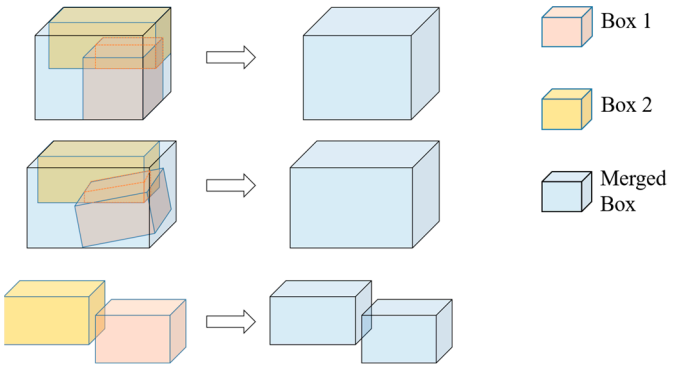
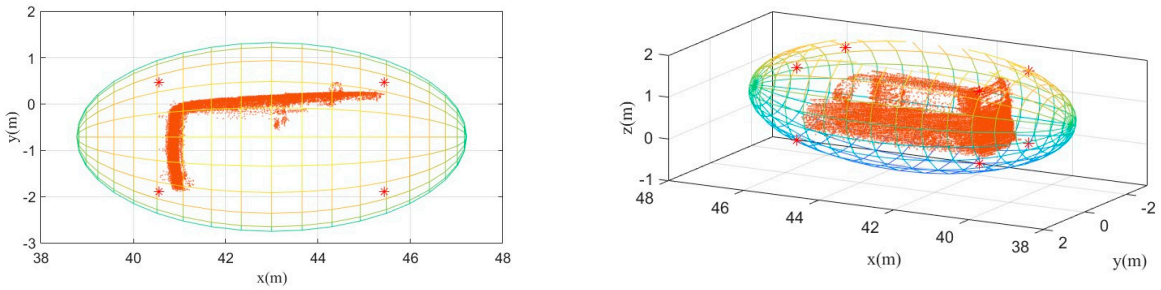


Figure 12. Schematic of the merging bounding box strategy.

Implement the merged bounding box strategy and the MVEE algorithm; cars can be enclosed completely, as Figure 13 shows.



(**a**) (**b**)

Figure 13. The Implementation of the algorithm to the car in (**a**) Top view and (**b**) 3-dimensional view. The red stars indicate the eight corner points of the merged bounding box, which are used in the MVEE algorithm to build the ellipsoid.

By storing all the information of the ellipsoids in the accumulated LiDAR map, the specific vehicle point clouds, which have been re-created behind the camera's FOV or initially generated before being identified as a vehicle in a single LiDAR frame, can still be removed successfully.

2.5. PCA for Edge Detection

Principal Component Analysis (PCA) is a method for reducing the dimensionality of data, simplifying the dataset while retaining its most important information. While

the context of a global map based on a point can contain billions of points and consume significant storage, PCA can be used to extract the essential characteristics of the environmental construction.

The basic PCA method analyzes the structure of the dataset by identifying its principal vectors. For 3-dimensional data points, there are three principal components. The values of these components depend on the covariance of the data points, and the principal vector represents the direction in which the data points are most widely distributed. Thus, along the third principal vector, the dataset has the smallest variance, meaning the points are most concentrated in this direction. In a 3-dimensional case, this PCA vector corresponds to the normal vector (as shown in Figure 14), and its value is defined as curvature. A larger curvature indicates that the surface tends to change more, suggesting the presence of edges. Consequently, applying PCA to the neighborhood of each point and using the curvature to detect edges in the map can build the contour for the body frame.

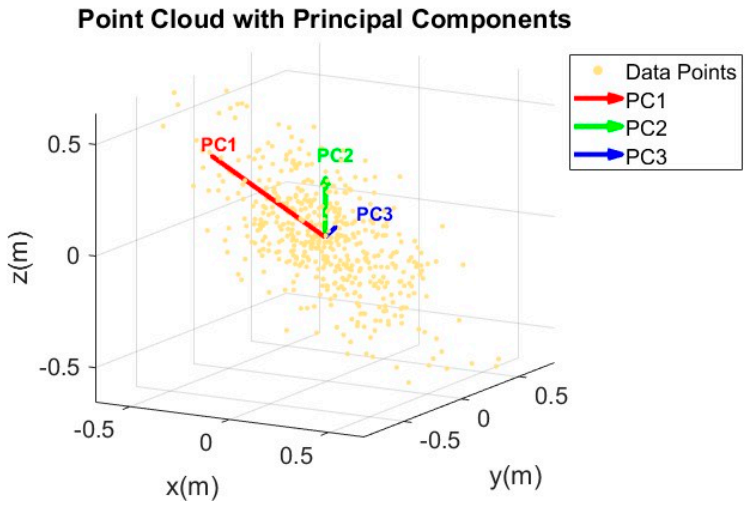


Figure 14. Schematic of the principal components.

Considering that point clouds become sparser as they are farther from the sensor, it was initially supposed this would affect the computation of curvature because the PCA method calculates curvature from the covariance of the neighbors. To verify this assumption, an experiment was conducted using a simulation to find the curvature at different distances for the same angle. The setup of the experiment is shown in Figure 15. By using the same number of LiDAR beams (represented by the black lines in the figure) to scan an edge wall at different distances (represented by the red lines in the figure), intersection points were identified, and curvature was calculated.

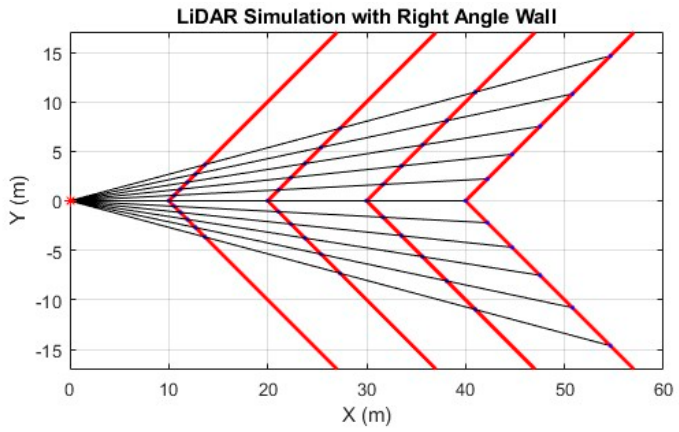


Figure 15. Demonstration of the experiment for testing the curvature of the angle at different distances.

As a result, it was found that the curvature remains consistent regardless of the distances between points. Even though the sparsity of the points changes significantly, the analysis results remain stable. This verifies the robustness of the curvature analysis method, indicating that detecting surface characteristics is independent of the sparsity of the point cloud data. Therefore, the algorithm can use a fixed curvature threshold regardless of the distance.

On the other hand, it was also found that for acute angles, the PCA method incorrectly identifies the normal vector as the first principal component, leading to inaccurate curvature analysis. However, acute angles are unlikely to be present in real-world scenarios, so this limitation of the algorithm can be ignored.

Having established the reliability of our curvature analysis, the focus now shifts to the parameters of the PCA-based edge detection algorithm. The PCA-based edge detection algorithm relies on two parameters: neighborhood size k and curvature threshold *Cur_thresh*, both affecting extraction accuracy. The *Cur_thresh* defines whether a point is classified as an edge; a lower threshold detects more edges but may include noise, while a higher threshold captures only sharp edges, potentially missing finer details. Similarly, larger k values smooth out details, making edges less distinct, while smaller k values capture finer changes but may be overly sensitive to noise. The selection of k and *Cur_thresh* depends on the specific point cloud density and characteristics. In our experiments, both k and *Cur_thresh* were selected empirically based on point cloud density. The k value is set to 40, and the *Cur_thresh* is set to 0.025. The detailed algorithm is shown as Algorithm 3.

Algorithm 3 PCA-based Edge Detection in Point Clouds

Input: Q (Data points matrix of size $N \times 3$), k (number of nearest neighbors)

Output: *edge point*

01: **For** each point i in **Q do**
02: Find k nearest neighbors of point i using KD-Tree
03: Extract neighbors' coordinate
04: Compute the centroid of the neighbors
05: Calculate covariance matrix Σ with dimension 3x3 w.r.t the neighbors
06: Perform SVD on covariance matrix Σ to get eigenvalues λ_1, λ_2, λ_3
07: Set $Curvature = \frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$
08: **If** $Curvature > Cur_thresh$ **do**
09: the point is defined as an *edge point*.
10: **End If**
11: **End For**

12: **Return** *edge point*

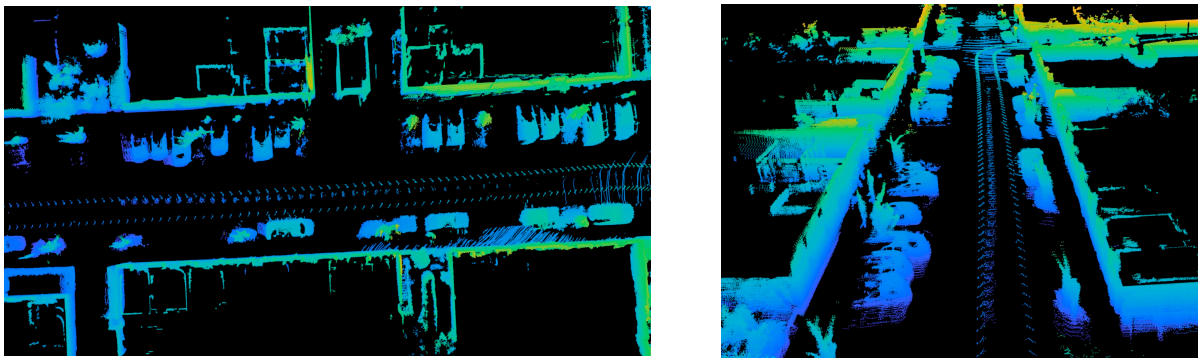
3. Experimental Results

In this section, the results of our proposed algorithm are presented, showcasing a step-by-step process leading to the final environmental contour maps. The experiment utilizes the most popular KITTI Dataset Sequence "00" to demonstrate the effectiveness of our approach in environmental modeling.

To illustrate the progression of our method, a series of images depicting each key stage of the process was shown:

A. High-Dense Point Cloud

Figure 16 illustrates the global map constructed from the original high-density point cloud. Initial pre-processing steps, including Statistical Outlier Removal (SOR) and basic ground removal, were applied to enhance the clarity of the environment.

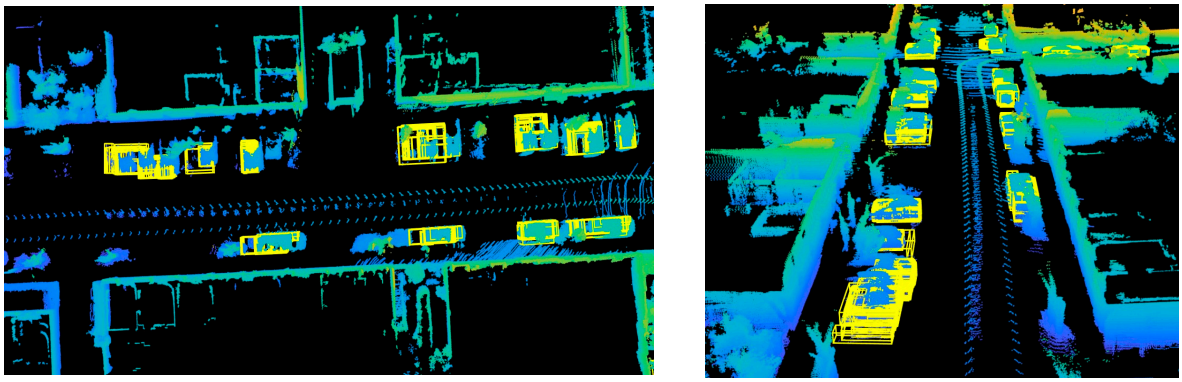


(**a**) (**b**)

Figure 16. Global map of high dense point cloud in (**a**) Top view; (**b**) 3-dimensional view.

B. Bounding Box Detection by Point Cloud-Image Fusion and Adaptive DBSCAN

Figure 17 demonstrates the bounding boxes of detected vehicles, defined by merging image recognition with point cloud clustering through Adaptive DBSCAN. Some vehicles are undetected due to differences in the fields of view (FOV) between the camera and LiDAR sensors.



(**a**) (**b**)

Figure 17. Bounding box based on image recognition and DBSCAN algorithm in the global map in (**a**) Top view and (**b**) 3-dimensional view.

To clarify the issue of missed vehicle detection within the LiDAR point cloud, the results of DBSCAN and image recognition are analyzed, as shown in Figure 18. In (a) Point Cloud and (b) Image of Frame 32, DBSCAN successfully detects the red car and black car on the left side and black car on the right side. Given the camera's FOV, no other vehicles are near the red and black cars, indicating that the missed bounding boxes likely belong to the cars behind the black car on the left side. This demonstrates that the camera's FOV plays a critical role in determining which vehicles are detected within the LiDAR point cloud. Therefore, the algorithm's ability to detect vehicles is directly limited by the camera's FOV, resulting in vehicles outside the FOV being completely omitted from the detection process.

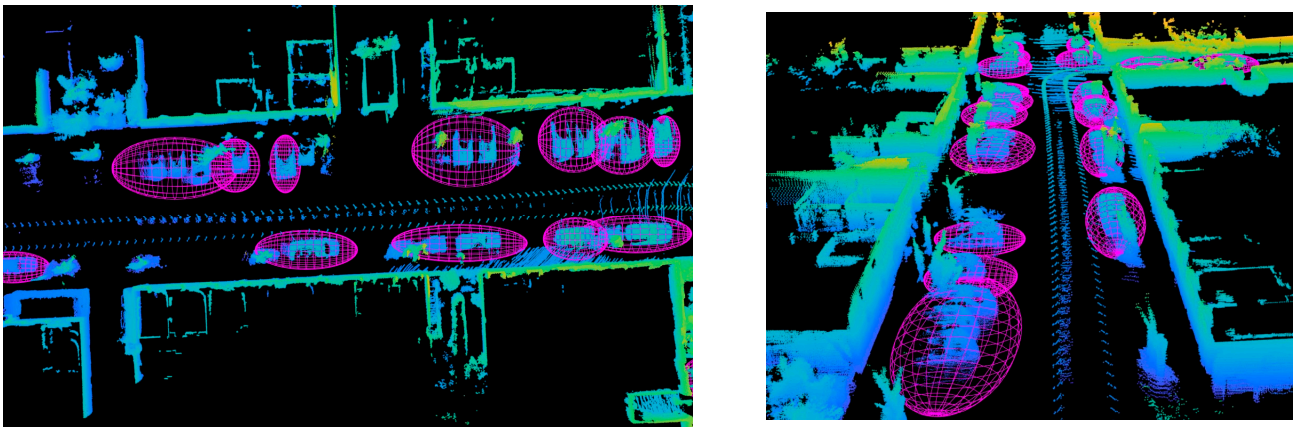
C. Minimum Volume Enclosing Ellipsoid

After applying the MVEE algorithm to the merged bounding boxes, one can obtain ellipsoids that enclose detected vehicles, as illustrated in Figure 19. The results demonstrate that the algorithm effectively captures the point cloud regions for most of the vehicles previously identified as objects. Due to the merging of intersecting bounding boxes, a single ellipsoid may sometimes enclose multiple vehicles rather than just one. While this can reduce the precision of object separation, it significantly minimizes storage requirements for the ellipsoid data, resulting in a more efficient overall process.



(**a**) (**b**)

Figure 18. The results of DBSCAN and image recognition. (**a**) Point Cloud and (**b**) Image of Frame 32.



(**a**) (**b**)

Figure 19. The result of applying MVEE for detected cars in (**a**) Top view and (**b**) 3-dimensional view. These ellipsoids represent the functionalized occupied region.

To validate the accuracy of our vehicle detection and removal algorithm based on the MVEE, a series of experiments using other scenarios from the KITTI dataset, which provides ground truth for vehicle tracking 3D bounding boxes, were conducted. In each experiment, the ground truth 3D bounding boxes were annotated, and the center of each box was determined. The MVEE function was then used to verify whether the centers of these ground truth bounding boxes fell within the ellipsoids generated by our algorithm. Detection accuracy was calculated as the percentage of ground truth bounding boxes with centers enclosed by the ellipsoids relative to the total number of ground truth bounding boxes.

The proposed algorithm was examined through two specific datasets: "2011_09_26_dri ve_0005" and "2011_09_26_drive_0009". The first dataset includes 160 frames of images and LiDAR point clouds with 12 annotated vehicles, achieving a detection accuracy of 74.54% with our algorithm. For the second dataset, containing 453 frames with 94 annotated vehicles, our method achieved an accuracy of 67.80%.

To understand and analyze the sources of the detection error, a detailed investigation using the KITTI tracking dataset, the vehicle recognition results by the YOLO model, and the projection of the LiDAR point cloud onto the corresponding images were conducted simultaneously. As shown in Figure 20, comparing Figure 20a,b reveals that a major source of error stems from the limitations of the applied YOLO model in vehicle recognition. Factors such as variations in lighting conditions and occlusions significantly impact detection accuracy. Vehicles partially obscured by other objects or shadowed regions are less likely to be accurately identified in the image. Moreover, Figure 20c highlights another prominent issue that arises in scenarios involving parking lots that are oriented perpendicular to the road. In such cases, vehicles parked at a distance may still appear within the camera's field of view (FOV) and can be detected in the image. However, the LiDAR's limited range

and resolution in distant regions often result in these vehicles being entirely or partially excluded from the LiDAR point cloud. Moreover, long distances between LiDAR and objects could lead to a low reflection rate and thereby cause empty reflections on these objects. Consequently, the clustering algorithm fails to identify these vehicles due to insufficient or incomplete LiDAR data. Briefly speaking, the main error sources come from the YOLO image recognition and the LiDAR hardware measurement physical limitations.



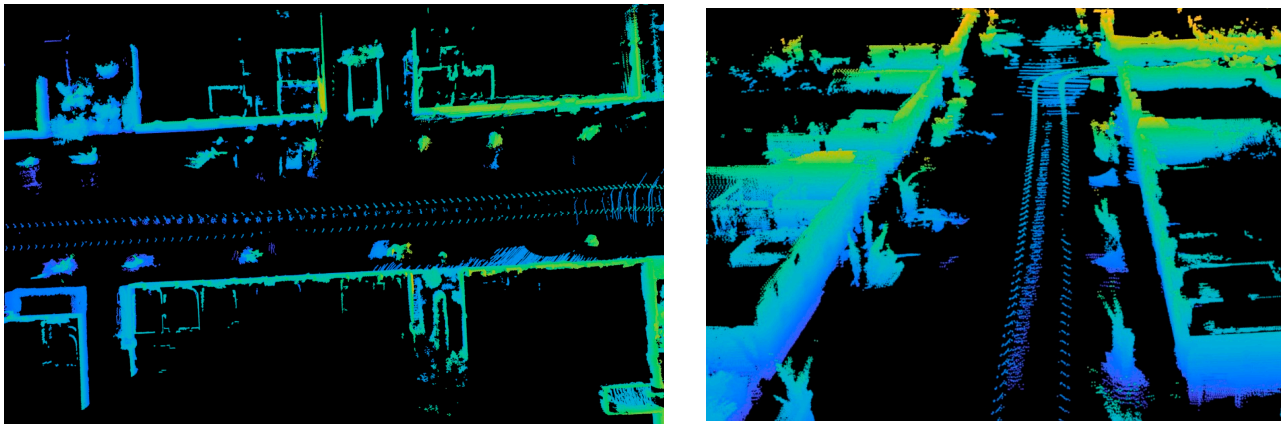
Figure 20. The comparison of the (**a**) KITTI tracking dataset with vehicle ground truth annotation (represented by green boxes), (**b**) the vehicle recognition results by the applied YOLO v8 model (with pink circles representing vehicles annotated by KITTI dataset but not detected by YOLO v8 model), and (**c**) the projection of the LiDAR point cloud onto the corresponding image (with pink circles indicating vehicles annotated by KITTI dataset but not projected on by the LiDAR point cloud).

These combined factors highlight the challenges posed by inconsistencies between image-based detection and LiDAR-based clustering. In particular, while the camera can detect vehicles based on pixel data at greater distances, the LiDAR system struggles to capture detailed point cloud information in these regions. This disparity affects the accuracy of the vehicle detection algorithm, resulting in misclassification or omission during vehicle filtering. Despite these limitations, the experiment verifies the reliability of our algorithm

to a certain extent. By accurately filtering out detected vehicles, our method enables cleaner contour extraction in subsequent processing stages, which is critical for downstream tasks.

D. Object Removal

Figure 21 illustrates the refined global map after removing vehicle points. Remaining only static environmental features such as trees, street lamps, and building walls. The figure clearly illustrates an environmental model with non-static objects effectively filtered out, providing a clearer representation of the scene.

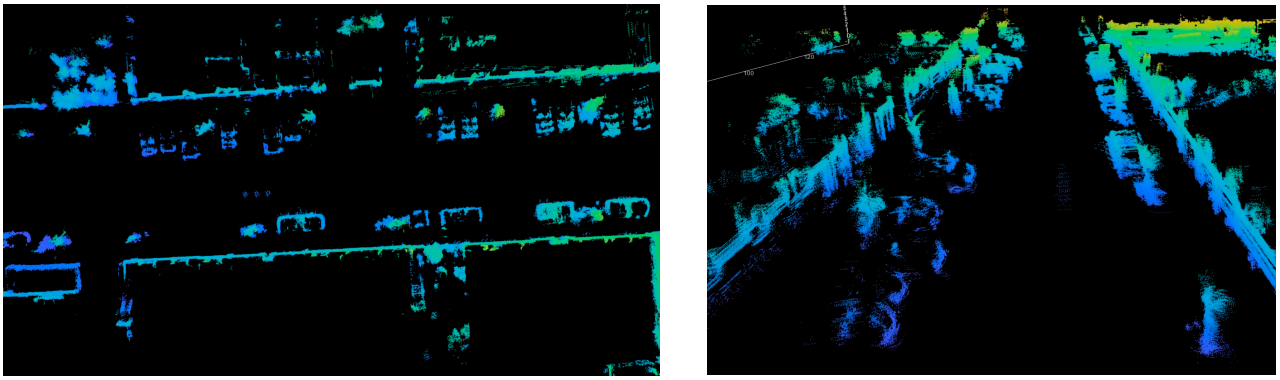


(**a**) (**b**)

Figure 21. Global map after removing the point clouds located in the ellipsoids: (**a**) Top view and (**b**) 3-dimensional view.

E. Environmental Edge Contour

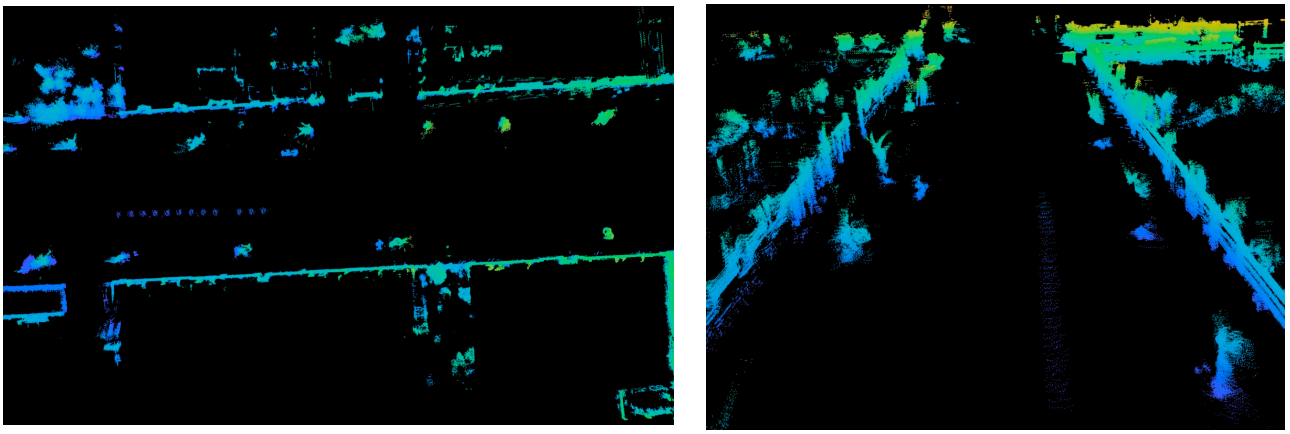
The edge of the environmental model was first extracted in the body frame and used to construct an initial global map after removing objects within each frame, as shown in Figure 22. However, due to discrepancies in the sensor FOVs, some vehicle points that were outside the camera's field of view during initial detection may still appear on the global map.



(**a**) (**b**)

Figure 22. Global contour map before global object removal in (**a**) Top view; (**b**) 3-dimensional view.

Finally, non-static objects tracked across multiple frames and modeled as MVEE are removed from the global map. The final results are presented in Figure 23, illustrating the refined global map after vehicles have been completely cleared.

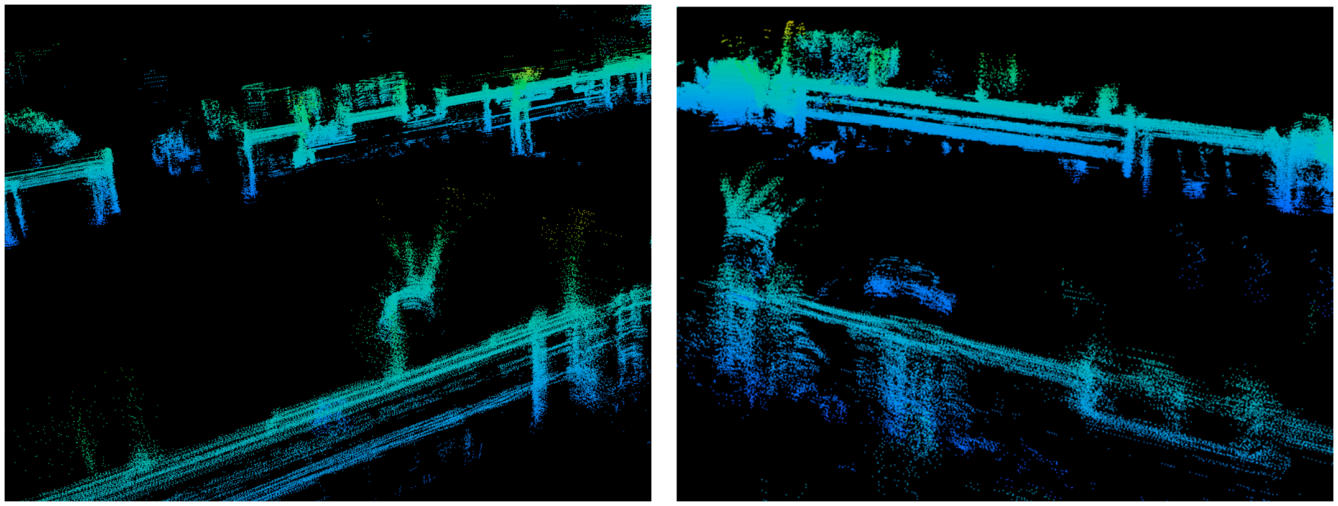


(**a**) (**b**)

Figure 23. Global contour map after global object removal (**a**) Top view; (**b**) 3-dimensional view.

This step successfully achieves the primary goal of our algorithm, which is to preserve the essential and recognizable features of the environment while minimizing storage overhead. The resulting environmental contour effectively captures critical structural elements, such as building outlines and road edges, ensuring that significant spatial information is retained without overburdening memory resources. By focusing on environmental contours, this approach not only reduces the data size but also enhances the clarity and interpretability of the 3D map.

Figure 24 presents a detailed close-up view of specific environmental features. The figure illustrates how our approach successfully delineates the fine edges and contours of the environment. Even when some structural characteristics are preserved within the remaining contours, the model maintains high fidelity in representing static environmental elements, ensuring an accurate portrayal of the scene.



(**a**) (**b**)

Figure 24. Detailed view of specific environmental contour features, where (**a**) and (**b**) are captured from different angle of view.

4. Conclusions and Future Work

In this paper, an environmental contour extraction algorithm based on LiDAR-Camera Fusion is proposed. The primary objective is to reduce the size of high-density, large-scale point clouds without compromising crucial environmental structural information, thereby significantly enhancing real-time processing efficiency and storage capacity. During the object detection phase, image recognition results are projected onto the 3D point cloud

through LiDAR-Camera fusion. The DBSCAN algorithm is then applied to cluster points within the bounding boxes. For the 3D bounding box fitting, this study employs the MVEE algorithm to ensure that unorganized vehicle points are completely enclosed, compensating for discrepancies in the FOV between the camera and LiDAR. In the edge extraction phase, PCA is applied to extract edges that represent the contours of environmental structures. Additionally, the potential to adjust the PCA algorithm based on the distance and angle of the points is explored. Experiments verify the feasibility and effectiveness of the proposed environment contour mapping strategy.

In contrast to AI-based methods that directly perform object detection and classification on LiDAR point clouds, the presented approach of object removal leverages image recognition results as a more straightforward and computationally efficient basis for detection. Although AI-based object detection in point clouds can achieve similar functions, it often requires labor-intensive training and complex annotation processes. For example, annotating vehicles in 3D point clouds requires manually selecting each object point-by-point, making it less intuitive compared to image-based recognition, which can rely solely on contours for detection. Moreover, applying AI models directly on point clouds involves handling sparse data and may require additional pre-processing, such as generating synthetic data or augmenting the dataset for model training.

However, a limitation of the proposed method is that only vehicles visible within the camera's FOV can be detected and processed by the proposed algorithm pipeline. This constraint means that vehicles occluded by other objects or partially outside the FOV may not be accurately removed in the final environmental map. The accuracy of detection may be affected in scenes with significant occlusions. Future work will focus on addressing the limitations of the current method by exploring multi-camera or panoramic cameras for long-range 3D LiDAR sensor fusion to alleviate occlusion issues, thereby improving the overall robustness of the system.

Author Contributions: Conceptualization, R.H. and C.-C.P.; methodology, T.-J.W. and C.-C.P.; software, T.-J.W.; validation, R.H. and T.-J.W.; formal analysis, C.-C.P.; investigation, R.H. and C.-C.P.; resources, C.-C.P.; data curation, T.-J.W. and R.H.; writing—original draft preparation, T.-J.W.; writing—review and editing, R.H. and C.-C.P.; visualization, T.-J.W.; supervision, C.-C.P.; project administration, C.-C.P.; funding acquisition, C.-C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science and Technology Council (NSTC), grant numbers 111-2923-E-006-004-MY3, 112-2221-E-006-104-MY3, and 113-2218-E-006-021.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Liang, L.; Ma, H.; Zhao, L.; Xie, X.; Hua, C.; Zhang, M.; Zhang, Y. Vehicle Detection Algorithms for Autonomous Driving: A Review. *Sensors* **2024**, *24*, 3088. [CrossRef] [PubMed]
2. Wang, Z.; Zhan, J.; Duan, C.; Guan, X.; Lu, P.; Yang, K. A Review of Vehicle Detection Techniques for Intelligent Vehicles. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *34*, 3811–3831. [CrossRef]
3. Jin, X.; Yang, H.; He, X.; Liu, G.; Yan, Z.; Wang, Q. Robust LiDAR-Based Vehicle Detection for On-Road Autonomous Driving. *Remote Sens.* **2023**, *15*, 3160. [CrossRef]
4. Golovinskiy, A.; Kim, V.G.; Funkhouser, T. Shape-based recognition of 3D point clouds in urban environments. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision (ICCV), Kyoto, Japan, 29 September–2 October 2009; pp. 2154–2161. [CrossRef]
5. Charles, R.Q.; Su, H.; Kaichun, M.; Guibas, L.J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 77–85. [CrossRef]
6. Zhou, Y.; Tuzel, O. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4490–4499. [CrossRef]

7. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
8. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587. [CrossRef]
9. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y. SSD: Single Shot MultiBox Detector. In *Computer Vision—ECCV 2016*; Springer: Cham, Switzerland, 2016; pp. 21–37.
10. Lin, T.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2999–3007. [CrossRef]
11. Cho, H.; Seo, Y.-W.; Kumar, B.V.K.V.; Rajkumar, R.R. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014.
12. Han, X.; Lu, J.; Tai, Y.; Zhao, C. A real-time LIDAR and vision based pedestrian detection system for unmanned ground vehicles. In Proceedings of the 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, Malaysia, 3–6 November 2015; pp. 635–639. [CrossRef]
13. Premebida, C.; Monteiro, G.; Nunes, U.; Peixoto, P. A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and Tracking. In Proceedings of the IEEE Intelligent Transportation Systems Conference, Seattle, WA, USA, 30 September–3 October 2007; pp. 1044–1049. [CrossRef]
14. Gonzalez, A.; Villalonga, G.; Xu, J.; Vazquez, D.; Amores, J.; Lopez, A.M. Multiview random forest of local experts combining RGB and LIDAR data for pedestrian detection. In Proceedings of the 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, Republic of Korea, 28 June–1 July 2015; pp. 356–361.
15. Xu, X.; Zhang, L.; Yang, J.; Cao, C.; Tan, Z.; Luo, M. Object Detection Based on Fusion of Sparse Point Cloud and Image Information. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 2512412. [CrossRef]
16. Liu, L.; He, J.; Ren, K.; Xiao, Z.; Hou, Y. A LiDAR–Camera Fusion 3D Object Detection Algorithm. *Information* **2022**, *13*, 169. [CrossRef]
17. Chen, X.; Ma, H.; Wan, J.; Li, B.; Xia, T. Multi-View 3D Object Detection Network for Autonomous Driving. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
18. Siwei, H.; Baolong, L. Review of Bounding Box Algorithm Based on 3D Point Cloud. *Int. J. Adv. Netw. Monit. Control.* **2021**, *6*, 18–23. [CrossRef]
19. Zand, M.; Etemad, A.; Greenspan, M. Oriented Bounding Boxes for Small and Freely Rotated Objects. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 4701715. [CrossRef]
20. Yi, J.; Wu, P.; Liu, B.; Huang, Q.; Qu, H.; Metaxas, D. Oriented Object Detection in Aerial Images with Box Boundary-Aware Vectors. In Proceedings of the 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 3–8 January 2021; pp. 2149–2158.
21. Feng, M.; Zhang, T.; Li, S.; Jin, G.; Xia, Y. An improved minimum bounding rectangle algorithm for regularized building boundary extraction from aerial LiDAR point clouds with partial occlusions. *Int. J. Remote Sens.* **2020**, *41*, 300–319. [CrossRef]
22. Naujoks, B.; Wuensche, H.-J. An Orientation Corrected Bounding Box Fit Based on the Convex Hull under Real Time Constraints. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1–6. [CrossRef]
23. Kwak, E.; Habib, A. Automatic representation and reconstruction of DBM from LiDAR data using Recursive Minimum Bounding Rectangle. *ISPRS J. Photogramm. Remote Sens.* **2014**, *93*, 171–191. [CrossRef]
24. Todd, M.J.; Yıldırım, E.A. On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discret. Appl. Math.* **2007**, *155*, 1731–1744. [CrossRef]
25. Bowman, N.; Heath, M.T. Computing minimum-volume enclosing ellipsoids. *Math. Program. Comput.* **2023**, *15*, 621–650. [CrossRef]
26. Borges, P.; Zlot, R.; Bosse, M.; Nuske, S.; Tews, A. Vision-based localization using an edge map extracted from 3D laser range data. In Proceedings of the 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010; pp. 4902–4909. [CrossRef]
27. Ceylan, D.; Mitra, N.J.; Li, H.; Weise, T.; Pauly, M. Factored Facade Acquisition using Symmetric Line Arrangements. *Comput. Graph. Forum* **2012**, *31 Pt 3*, 671–680. [CrossRef]
28. Tsai, T.-C.; Peng, C.-C. Ground segmentation based point cloud feature extraction for 3D LiDAR SLAM enhancement. *Measurement* **2024**, *236*, 114890. [CrossRef]
29. Jutzi, B.; Gross, H. Nearest neighbour classification on laser point clouds to gain object structures from buildings. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2009**, *38*, 4–7.
30. Pauly, M.; Gross, M.; Kobbelt, L.P. Efficient simplification of point-sampled surfaces. In Proceedings of the IEEE Visualization (VIS 2002), Boston, MA, USA, 27 October–1 November 2002; pp. 163–170. [CrossRef]
31. Huang, H.; Li, D.; Zhang, H.; Ascher, U.; Cohen-Or, D. Consolidation of unorganized point clouds for surface reconstruction. *ACM Trans. Graph.* **2009**, *28*, 1–7. [CrossRef]
32. Chuang, C.-S.; Peng, C.-C. Development of an Uneven Terrain Decision-Aid Landing System for Fixed-Wing Aircraft Based on Computer Vision. *Electronics* **2024**, *13*, 1946. [CrossRef]

33. Kuçak, R.A. The Feature Extraction from Point Clouds using Geometric Features and RANSAC Algorithm. *Adv. LiDAR* **2022**, *2*, 15–20.
34. Ji, C.; Li, Y.; Fan, J.; Lan, S. A Novel Simplification Method for 3D Geometric Point Cloud Based on the Importance of Point. *IEEE Access* **2019**, *7*, 129029–129042. [CrossRef]
35. Chen, X.; Liu, Q.; Yu, K. A Point Cloud Feature Regularization Method by Fusing Judge Criterion of Field Force. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 2994–3006. [CrossRef]
36. Zang, Y.; Chen, B.; Xia, Y.; Guo, H.; Yang, Y.; Liu, W.; Wang, C.; Li, J. LCE-NET: Contour Extraction for Large-Scale 3-D Point Clouds. *IEEE Trans. Geosci. Remote Sens.* **2023**, *61*, 5704413. [CrossRef]
37. Redmon, J.; Divvala, S.K.; Girshick, R.B.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
38. Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD-96 Proc.* **1996**, *96*, 226–231.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.