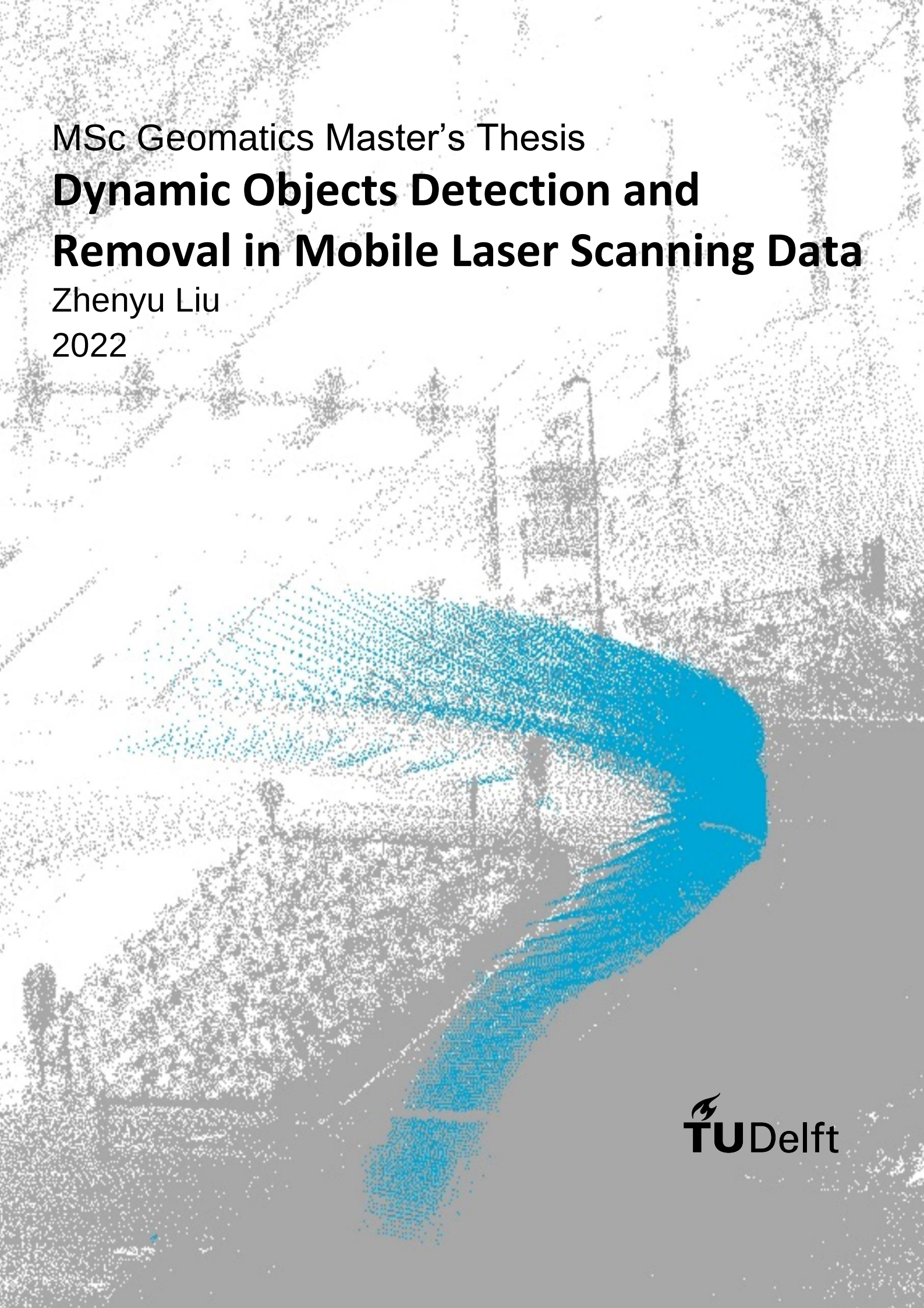
MSc Geomatics Master's Thesis

Dynamic Objects Detection and Removal in Mobile Laser Scanning Data

Zhenyu Liu

2022



TU Delft

Dynamic Objects Detection and Removal in Mobile Laser Scanning Data

By

Zhenyu Liu

in partial fulfilment of the requirements for the degree of

Master of Science
in Geomatics

at the Delft University of Technology,
to be defended publicly on Monday, June 27, 2022 at 8:45 AM.

Supervisor:	Prof. dr. ir. P.J.M. van Oosterom,	TU Delft
	Dr. Jesús Balado Frías,	TU Delft
External Mentors:	Dr. ir. Bart Beers,	CycloMedia
	MSc. Arjen Swart,	CycloMedia
Thesis committee:	Drs. D.J. Dubbeling,	TU Delft
	MSc. Mieke Kuschnerus,	TU Delft

This thesis is not confidential and will be made public on June 28, 2022.

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Contents

Abstract

Many MLS point cloud application scenarios, such as navigation and localization algorithms, require only static environments, but the original MLS data usually inevitably includes many dynamic objects such as moving vehicles, bicycles, and pedestrians. Therefore, these dynamic objects need to be removed before using MLS point clouds. This thesis designs an efficient and memory-friendly Octomap-based dynamic object detection and removal method for MLS data. Firstly, the original MLS data is split into multiple data frames based on the timestamp of each capture point. Each data frame is inserted into a separate Octomap along with its neighboring data frames. The free points in all Octomaps are extracted by setting an occupancy probability threshold. Second, the region of interest (ROI) related to the dynamic object is delineated by the MLS sensor mounting height and the local large vehicle height limit. Only the free points located within the ROI are retained. Then the free-point rate and the multi-return rate are calculated for each free point using a fixed radius spatial search to denoise and detect vegetation points. Finally, the KNN spatial search is used to remove vegetation points and extract dynamic objects from the free points. The proposed method is tested in four case sites in Delft, the Netherlands and its producer's and user's weighted average dynamic object detection and extraction accuracies are 88.004% and 82.624%, respectively. The weighted average overall accuracy is 99.833%. Compared with the original Octomap, the proposed method is 35.472% more efficient on average and can be further accelerated by parallel computing, with a maximum memory consumption of only 42.437% of the original Octomap. The implementation results and accuracy assessment demonstrate that the proposed method can be effectively applied to dynamic object detection and extraction tasks in MLS data sets in a compute-friendly and memory-friendly way.

Acknowledgements

As my master's project is nearing completion, I first want to express my highest gratitude to my supervisors, **Peter** and **Jesus**, for introducing me to the field of point clouds and inspiring my strong interest in research. Their academic guidance had a profound impact on me and encouraged me to continue pursuing my academic dreams after my master's program.

I am very grateful to **Arjen** and **Bart** from CycloMedia for giving me a lot of data and technical support. Their comments and ideas helped me to finalize my dissertation topic as dynamic object processing in MLS data. I am thankful to **Mieke**, the co-reader of this thesis, for her academic advice. I also thank to thank **Dirk** for hosting each phase meeting during my graduation program.

Finally, I would like to thank my **parents**. Although you are far away on the other side of the world, your support and encouragement have pushed me to bravely overcome one academic and life challenge after another during these two tough years.

Acronyms

2D: Two-dimensional

3D: Three-dimensional

AABB tree: Axis Aligned Bounding Box tree

ALS: Airborne Laser Scanning

ASM: Active Shape Model

BLS: Backpack Laser Scanning

CPU: Central Processing Unit

CUDA: Compute Unified Device Architecture

DBMS: Database Management Systems

FCN: Fully Convolutional Neural Network

FOV: Field of View

GNSS: Global Navigation Satellite System

GPU: Graphics Processing Unit

HD-map: High-Definition Map

HMLS: Handled Mobile Laser Scanning

IMU: Inertial Measurement Unit

KNN: K-nearest Neighbors

LiDAR: Light Detection and Ranging

LRF: Laser Range Finder element

MLS: Mobile Laser Scanning

PCL: Point Cloud Library

RANSAC: Random Sample Consensus

ROI: Region of Interest

SHOT: Signature of Histograms of Orientations

SLAM: Simultaneous Localization and Mapping

SVM: Support Vector Machines

TEN: Tetrahedral Network

TIN: Triangular Irregular Networks

TLS: Terrestrial Laser Scanning

1. Introduction

This chapter first overviews the background information of the dynamic object problem in Mobile Laser Scanning (MLS) data and explains the importance of solving this problem for MLS point cloud data applications (Section 1.1), then defines the research scope in Section 1.2, and finally introduces the structure of this thesis in Section 1.3.

1.1 Background and Motivation

LiDAR technology provides a revolutionary and efficient way to capture 3D spatial data with high geometric accuracy and rich detail in the real world (Che et al., 2019). According to different carrying platforms of Light Detection and Ranging (LiDAR) sensors, there are three types of mainstream acquisition methods of point cloud data, including MLS, Airborne Laser Scanning (ALS), and Terrestrial Laser Scanning (TLS). MLS systems are usually mounted on land-based mobile platforms such as vehicles. ALS systems are often deployed on aircraft. TLS sensors are usually mounted on static tripods to scan the surrounding area (Hyyppä et al., 2013).

Compared with other acquisition methods, MLS has its unique advantages: It can capture the data with better visibility, accuracy, and resolution than ALS and has higher collection efficiency than static TLS (Williams et al., 2013). These advantages make MLS fit the demand for point clouds data in urban scenes, especially in linear road environments (Soilán et al., 2019). So in recent years, MLS has been widely used in many urban applications such as urban land cover analysis, urban environment monitoring, digital 3D urban modeling, and self-driving vehicles (Di Stefano et al., 2021; Y. Wang et al., 2019). MLS data is usually integrated with other sensor data to get richer information for their projects in many research and commercial applications. For example, one common sensor combination is LiDAR, Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), and the RGB camera (see Figure 1), which generates accurate data timestamps and sensor movement trajectories while collecting MLS data sets. The spatiotemporal information brought by these additional sensors often assists MLS data for higher quality and more efficient environmental reconstruction (Čerňava et al., 2019; Rodríguez-Cuenca et al., 2015; Williams et al., 2013).

However, some problems with the original MLS data set often causing limitations in several application scenarios. One of the most common problems is dynamic objects. According to the motion state of objects during scanning, the environment objects in MLS data can be

divided into static objects and dynamic objects. Common static objects include roads, buildings, trees, and parked vehicles in urban environments. Pedestrians and moving vehicles are the main dynamic objects (see Figure 1). Most application scenarios of MLS data, like object extraction, change detection, and generating HD-maps to support navigation and location services, require only static environment objects (Balado et al., 2019; L. Ma et al., 2018). However, due to the data acquisition method of MLS, it is impossible to completely avoid dynamic objects in the original point cloud data, especially in areas with large human and vehicle flows in cities. On the other hand, the moving route and speed of the MLS sensor are restricted by traffic laws, road networks, and other factors (Balado et al., 2020). Therefore, the MLS sensor and its nearby dynamic objects (mainly moving vehicles) are highly likely to be in the same or opposite moving directions at similar speeds. A more serious problem arises in the captured point cloud data if dynamic objects are moving in the same direction as the MLS sensor. These dynamic objects accompanying the MLS sensor are continuously scanned, which makes them very seriously stretched in the collected point cloud data (see Figure 2). In some studies, this phenomenon is named the ghost trail effect (Pagad et al., 2020; Pomerleau et al., 2014). So, the problem of dynamic objects in MLS data is more severe and complex than in other types of point cloud data.

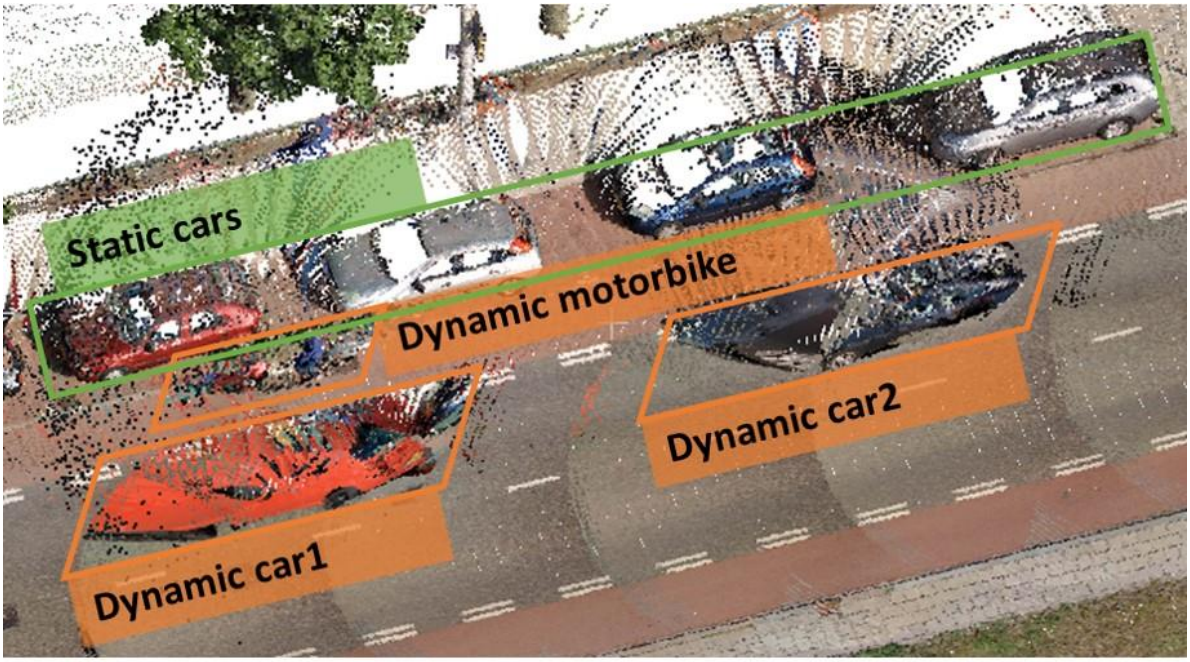


Figure 1: A dynamic motorbike, dynamic cars, and static cars in MLS data

Thus, distinguishing and removing dynamic objects from static objects is important in MLS data preprocessing in many application tasks, such as localization and navigation. The performance of dynamic object detection and removal methods can directly affect the quality

of subsequent MLS applications. If dynamic objects cannot be removed accurately, many point cloud application algorithms may not work properly. For example, residual dynamic objects reduce the location accuracy of point-based HD-map (Endo et al., 2021; Wen et al., 2021). However, MLS sensors' speed and moving direction are constantly changing, making their relative motion with the surrounding objects more complex. Many detection methods based on ALS and TLS data cannot be directly applied to MLS data. So it is more challenging to accurately detect dynamic objects in MLS data than in TLS and ALS data.

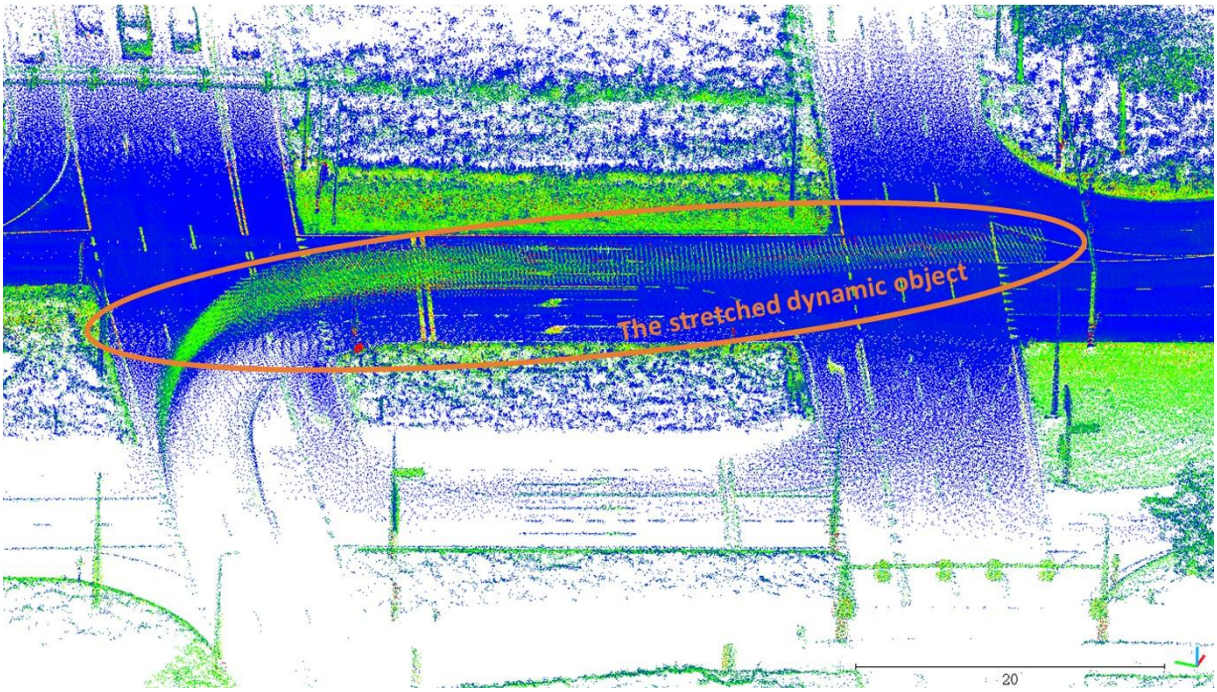


Figure 2: The stretched dynamic object in MLS data

To address the problems and challenges caused by dynamic objects in MLS data, the research aims to design a dynamic object detection and removal method based on MLS data, which can be accelerated with parallel computing. The proposed method first generates an Octomap to extract all free points from the MLS data sets. Free points in this thesis refer to all points located in the Octomap space whose occupancy probability is less than a given probability threshold and usually has a higher probability of being part of a dynamic object (See Section 3.1 and Section 4.1 for a more detailed definition of free points and how it differs from dynamic points). Then the trajectory of the MLS sensor is used to delimitate the Region of Interest (ROI) from all extracted free points. The measurement noise is filtered from the free points located in the ROI. Alter that the number of returned LiDAR rays is used to remove the vegetation from the remaining free points. Finally, dynamic objects are extracted and removed by using spatial search with filtered free points.

1.2 Research Questions

The main research question of this thesis is as follows:

"How to detect and remove the dynamic objects from the MLS data?"

From the perspective of data, this problem can be expressed as to how to return a point cloud without dynamic objects after giving an MLS data and its corresponding sensor movement trajectory?

After defining the main question of this research, some sub-questions are derived from it:

- How to detect and remove dynamic objects and avoid residue?
- How to avoid detecting and removing static environment objects?
- What factors affect the detection results?
- How to use MLS sensor trajectory to assist detection and removal operations?
- What types of objects often lead to misdetection?
- How to improve the computational efficiency for large-scale data?

1.3 Research Scope

This section is intended to clarify the research scope to help the research focus better on the core research issues defined in Section 1.2.

Firstly, the point cloud collection method of this research is MLS. Some studies interpret MLS as all LiDAR systems mounted on land-based mobile platforms (including humans), so the backpack-MLS (BLS) and handled-MLS (HMLS) are also considered a type of MLS in their studies (Hauser et al., 2016). Although there are many similarities between BLS and vehicle-based MLS, there are still many significant differences between these two systems in sensor height and movement speed, which make the point density and visibility of the collected data different. This research mainly focuses on object detection in an outdoor environment, such as roads. The vehicle-based MLS is the main way of data collection in a large range (such as city-scale), so MLS in this study refers specifically to vehicle-based MLS.

Second, this research focuses on dynamic objects. In the real world, dynamic objects exist from underwater to high in the sky. Some birds in flight are indeed inadvertently scanned

during actual data collection. However, this research only focuses on ground moving objects, such as cyclists, vehicles, etc. This is also an important prerequisite for reducing the detection region in Section 4.2. In addition, vehicles that are temporarily stopped during the whole scanning process due to traffic signals or other reasons are not considered dynamic objects. Valid dynamic objects in this study refer to objects that are moving during the whole or part of the scanning period, including vehicles that suddenly start or stop during the scanning process.

Third, the point cloud processing operation concerned in this research is detecting and removing dynamic objects. Although point cloud processing operations such as ground extraction and vegetation extraction are also used in the intermediate steps of the proposed method. However, due to the relatively limited research time, the performance of these methods is not included in the core research questions mentioned in Section 1.2. However, it must be acknowledged that the performance of these operations have an impact on the final result. Therefore, this thesis will analyze which wrongly detected and removed objects are caused by bad ground extraction and vegetation extraction results based on the implementation results in Chapter 5 and Chapter 6. These two operations can also be replaced with other better-performance ground and vegetation extraction methods. This does not affect the overall workflow of the proposed method.

1.4 Thesis Structure

The rest of this thesis is organized as follows:

- Chapter 2 reviews previous related research about dynamic object detection and change detection in point cloud data, including single-frame data methods and multi-frame data methods.
- Chapter 3 shows the theoretical background of Octomap and neighborhood query, which are the key concepts involved in this research methodology.
- Chapter 4 presents the proposed methodology of dynamic object detection and removal. It describes the five main sub-steps of the methodology, including extracting the free points, reducing the detection area, removing outliers, removing vegetation, and extracting dynamic objects in detail.
- Chapter 5 indicates the implementation details and discusses the running time and the memory consumption of the acceleration method used in implementation.

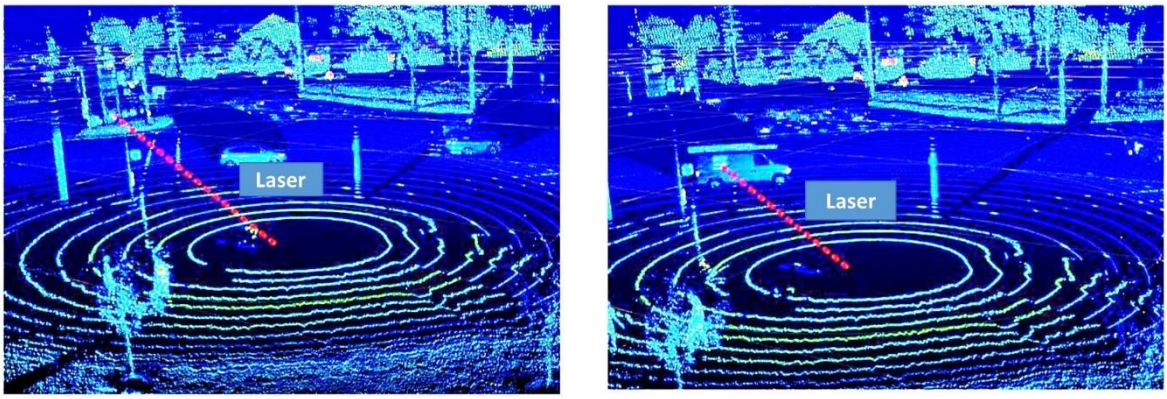
- Chapter 6 presents the final implementation results and is devoted to evaluating the final results and discussing the performance of the proposed methods.
- Finally, Chapter 7 gives conclusions and future works of this research.

2. Related Work

This chapter summarizes the recent research on dynamic object detection and change detection in LiDAR data. The detection and removal of point cloud dynamic objects are of great significance in many scenes, such as autonomous driving (Mekala et al., 2021), 3D point cloud mapping (Arora et al., 2021), and environmental monitoring (Okyay et al., 2019; Teo & Shih, 2013), so many relevant studies have been done in academia. The current methods can be divided into single-frame data and multi-frame data methods according to whether multiple scans (or continuous scans in MLS) are required, which are respectively introduced in Section 2.1 and Section 2.2. Finally, Section 2.3 summarizes the relevant research on this issue.

2.1 Single-frame Data Methods

The single-frame lidar data refers to the scan data obtained by the sensor in an instant or a very short period. ALS and TLS scan results are usually single-frame data. For MLS, the data obtained by a single rotation of the sensor (360°) is also considered single-frame data.



(a) Create the prior map for the environment

(b) Detect the dynamic object using background subtraction

Figure 3: The dynamic object detection based on the background subtraction (T. Zhang & Jin, 2022)

One of the simplest single-frame data methods is to use a prior map for change detection. The dynamic object detection is formulated as a background subtraction problem if the prior map is prepared in advance. All objects that do not exist in the prior map are considered dynamic objects (Kiran et al., 2019). Figure 3 indicates the main idea of a dynamic object detection method based on background subtraction: Given a static background, the LiDAR

sensor should be able to see the background points closest to it in any direction, so lines of sight are created between them (Figure 3. A). If these lines of sight are obscured by an object from the real-time scan data, this object is dynamic (Figure 3. B). The representation forms of prior point cloud maps are diversified. In addition to the original point cloud data, it can also be based on the semantic point cloud map (W.-C. Ma et al., 2019), probability occupancy grid (Anderson-Sprecher et al., 2011), or feature map extracted from the original point cloud (Yin et al., 2020).

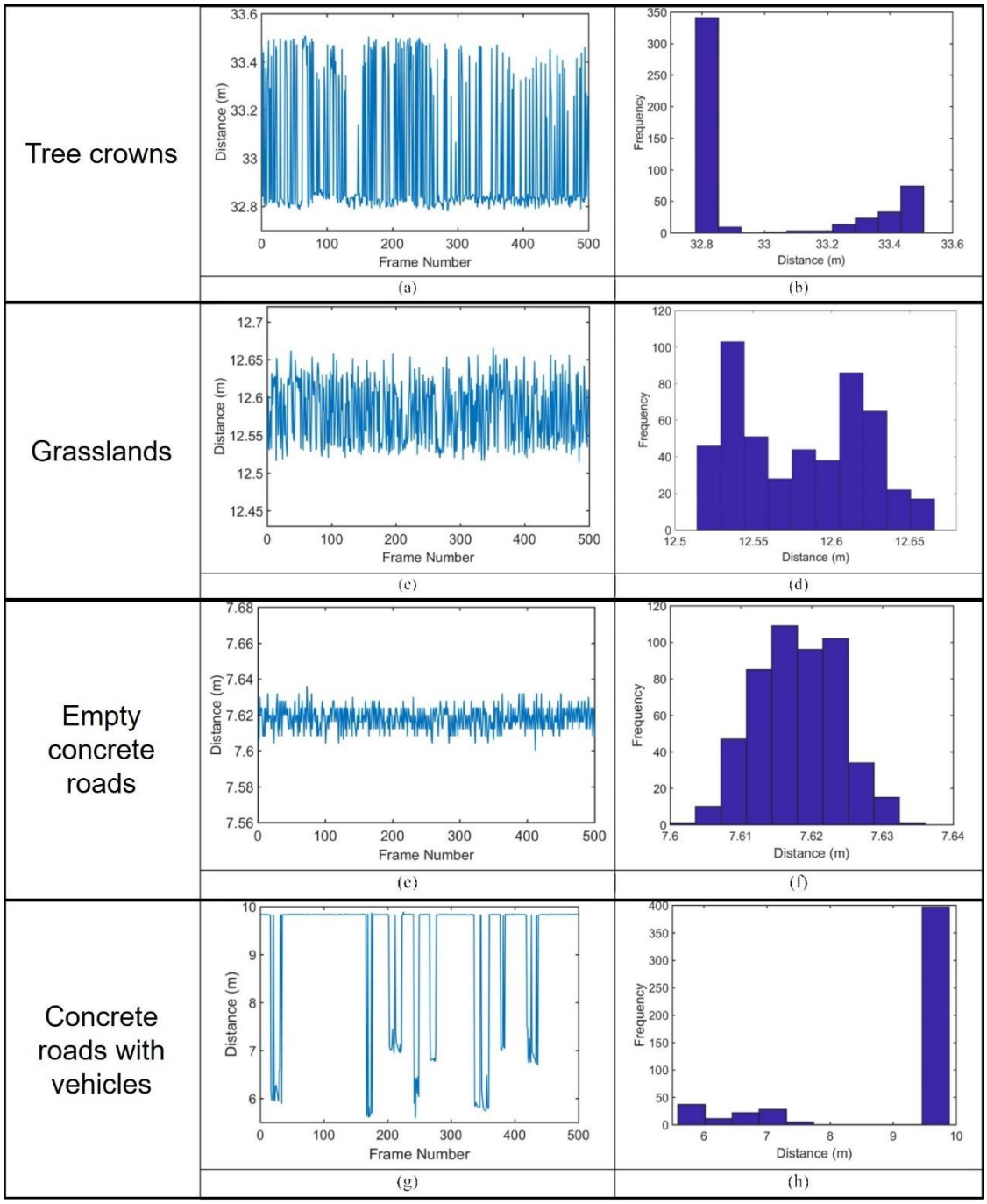


Figure 4: Relative distances and corresponding histograms of different background objects (Xia et al., 2022).

However, it is not easy to construct a prior map in practice. The first issue is that the generation and updating of the prior map also need to remove dynamic objects. So prior map construction and dynamic object detection are chicken-and-egg problems due to their interwoven nature in this method (Kim & Kim, 2020). Another problem is that not all environment objects are completely static. It is difficult to accurately represent dynamic background objects such as tree crowns and grasslands in the prior map because they are not rigid. Figure 4 illustrates the time series of the relative radial distances between the LiDAR sensor and different kinds of background objects and their corresponding histograms to further indicate this problem. It indicates the difference in the radial distance distributions between the dynamic background objects and static background objects. The distance of dynamic background objects such as the tree crowns (Figure 4. (a) and (b)) and grasslands (Figure 4. (c) and (d)), fluctuate sharply. The distance of static background objects like empty concrete road surface (Figure 4. (e) and (f)) slightly fluctuate unless there are vehicles on this road (Figure 4. (g) and (h)).

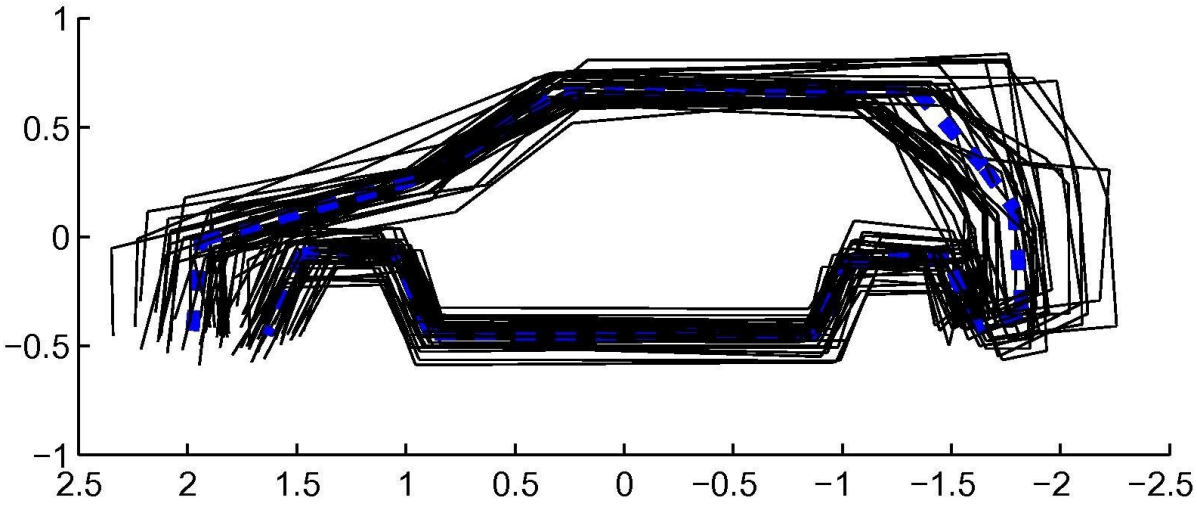


Figure 5: The ASM alignment of vehicles, mean of samples in blue dashed line (Xiao et al., 2016).

Another idea based on prior knowledge is to extract potential moving objects using feature or model matching (Cheng et al., 2014; Ding & Wang, 2021). Common features include object dimension (length, width, and height), volumetric features (object surface area, vertical projected area, and volume), relative position (maximum relative height and mean relative height), and vertical point distribution histogram, etc. (Xiao et al., 2016). In addition to defining features for dynamic objects, these target objects can also be modeled directly as Active Shape Model (ASM) (Zeeshan Zia et al., 2013). An example of vehicle ASM is shown in Figure 5 and the extracted results of this model are shown in Figure 6. Then, based on

these models and features, dynamic objects are extracted by classifiers, like Support Vector Machines (SVM) (Y.-W. Chen & Lin, 2006) or random forest (Breiman, 2001).

Feature-based and model-based approaches also face some problems. First, in addition to vehicles, common dynamic objects also include cyclists and pedestrians. Even just focusing on vehicles, the differences between different types of vehicles are very large. Most of the previous methods only focused on small vehicles and did not include large vehicles such as large trucks or buses. Some cities also have special vehicles such as ground rail vehicles and tricycles. To be able to cover various types of dynamic objects, some studies chose to increase the number of features. For example, Lin et al. (2018) raised a 26-dimensional feature and Guo et al. (2019) raised a 32-dimensional feature to extract vehicles, cyclists, and pedestrians. Iqbal et al. (2021) used a transfer learning strategy to extract 128 features from the original point cloud. In general, researchers tend to use as many features as possible when they lack relevant prior knowledge, which leads to a significant increase in computing time and memory requirements (Weinmann et al., 2015). The second problem is that even when appropriate features or models are obtained, they are generally only applied to static or low-speed objects. As shown in Figure 2, the ghost trail effect makes objects which move at high speed cannot match with the model or features of static objects. Depending on speed and trajectory (such as moving straight, changing lanes, or turning at the corner), the same dynamic object takes on a completely different geometrical shape. Therefore, it is difficult to define suitable features and models for dynamic objects.

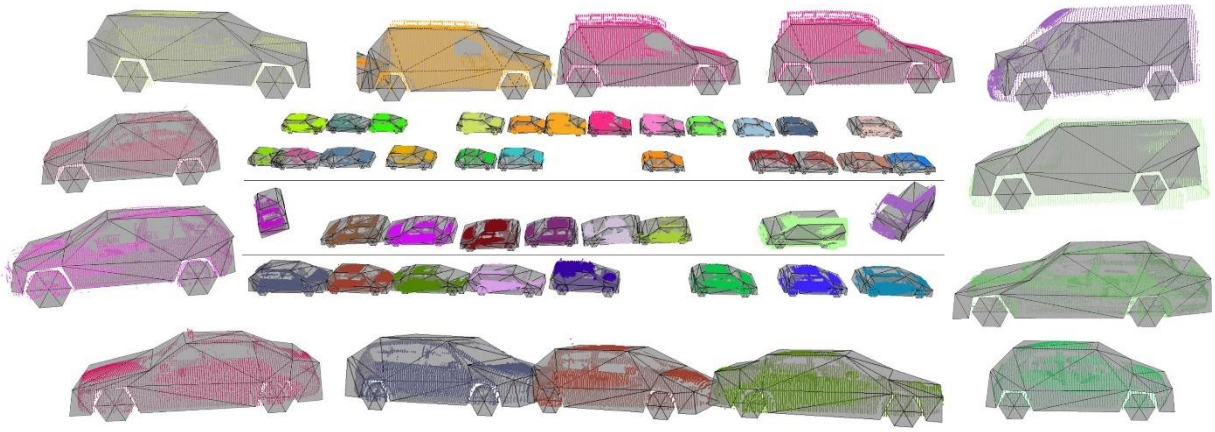


Figure 6: The vehicle ASM fitting examples (Xiao et al., 2016).

Some studies attempt to extract the motion state of objects from a single-frame point cloud data, although a single-frame data is not generally considered to contain enough motion information. Yao et al. (2011) detected moving vehicles in ALS data using motion artifacts

caused by ALS sensors moving relative to the static environment. This method approximates a static vehicle as a rectangle from a 2D perspective. For moving vehicles, this rectangle will be stretched into a parallelogram. The shearing angle θ_{SA} of this parallelogram is determined by the angle θ_v between the movement direction of the vehicle and the ALS sensor. So a motion artifacts model illustrated in Figure 7 infers the dynamic state of vehicles based on the relative relation between the original rectangle and the stretched parallelogram. Given the sensed length (l_s) and original length (l_v) or the original aspect ratio (Ar) and the sensed aspect ratio (Ar_s) of a vehicle in the ALS data. And the velocity of the ALS sensor (v_L) and angle θ_v are also known. The velocity of the vehicle (v) is calculated in Eq.1 (l_s and l_v can be replaced by Ar_s and Ar). Then shearing angle θ_{SA} used to compare the velocity of the vehicle (v) is obtained in Eq.2. So the dynamic state of the vehicle is recovered with v and θ_{SA}.

$$l_s = \frac{l_v \cdot v_L}{v_L - v \cdot \cos(\theta_v)} = \frac{l_v}{1 - \frac{v}{v_L} \cdot \cos(\theta_v)} \tag{1}$$

$$\theta_{SA} = \arctan\left(\frac{v \cdot \sin(A)}{v_L - v \cdot \cos(A)}\right) + 90° \tag{2}$$

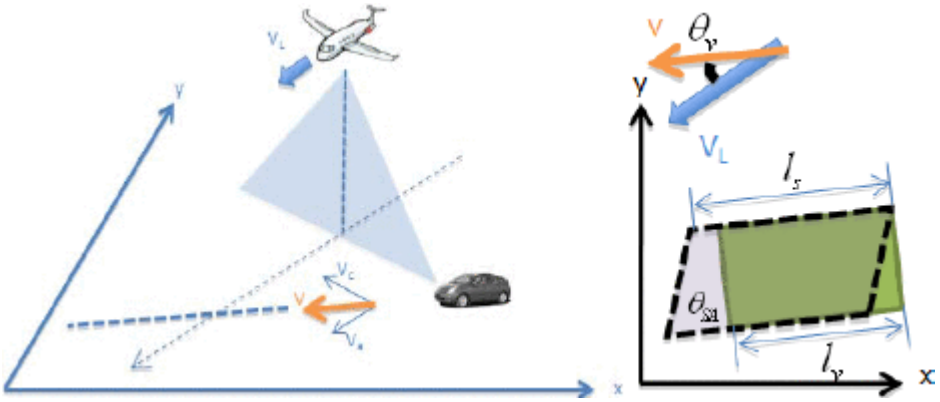


Figure 7: Motion artifacts model (Yao et al., 2010).

Although the motion artifacts model has good results when dealing with vehicle objects in ALS, they are not suitable for pedestrians or cyclists because these two kinds of objects do not fit well with rectangles and parallelograms. Another problem is that because the ALS sensor moves so fast, the scanning time of an area is very short. Therefore, it is easy to capture the instantaneous motion state of the moving object. By contrast, MLS sensors are limited by the speed of the data acquisition vehicle and traffic rules and spend more time scanning the same area. As a result, it is possible to capture the moving state of a dynamic vehicle for a long time, resulting in the captured object not moving in a fixed direction, but (continuously) changing the direction of motion during the scan (Figure 2). So it is difficult to

simply generalize the captured dynamic object as a parallelogram and generate the motion artifacts model.

Since the advent of methods such as PointNet (Charles et al., 2017) and VoxelNet (Zhou & Tuzel, 2018) in recent years, deep learning has been widely applied to the object detection tasks in point cloud data (Ku et al., 2018; Shi et al., 2019). Most of these methods are directly applied to the single-frame point cloud data, and show excellent performance in pedestrian, cyclist, and vehicle detection tasks. However, the good performance of such methods usually requires the support of a high-quality training set. For dynamic object detection tasks, although some researchers have made the related data set (Pfreundschuh et al., 2021), the relevant high-quality available data sets are still not enough in general. For large open data sets such as KITTI (Geiger et al., 2013), training samples are often very imbalanced, especially for uncommon moving objects. This may affect the quality of the results of multi-type object recognition tasks (Wu et al., 2021).

2.2 Multi-frame Data Methods

Multi-frame data is a collection of multiple single-frame data, such as the continuously scanning MLS data is a typical multi-frame data. Earlier research simply interpreted multi-frame data as a single frame with more points (Sun et al., 2020) but ignored the relative relationship between each data frame. Subsequent studies proved that spatiotemporal correlations among consecutive frames provide much useful information especially for detection of dynamic objects (Huang et al., 2020; Luo et al., 2018).

A common multi-frame method is object tracking. This method thinks that multi-frame data is a time series of single-frame data, so it starts from the first frame data and takes the data of the current frame as the reference of the data of the next frame to find the object with position change in the next frame. There were many model-based object tracking approaches in early research (Petrovskaya & Thrun, 2009; Shackleton et al., 2010). They usually apply model-based object detection on each single-frame data (see Figure 8) and then use methods such as Kalman Filters (Zhao & Thorpe, 1998) or Signature of Histograms of Orientations (SHOT) descriptors (Tombari et al., 2010) to match the same object in different data frames. If the position of the detected object changes, this object is dynamic. Such methods usually require prior knowledge of the object to be detected and are therefore very effective when performing tracking tasks for specific targets, such as the pedestrian detection in Figure 8.

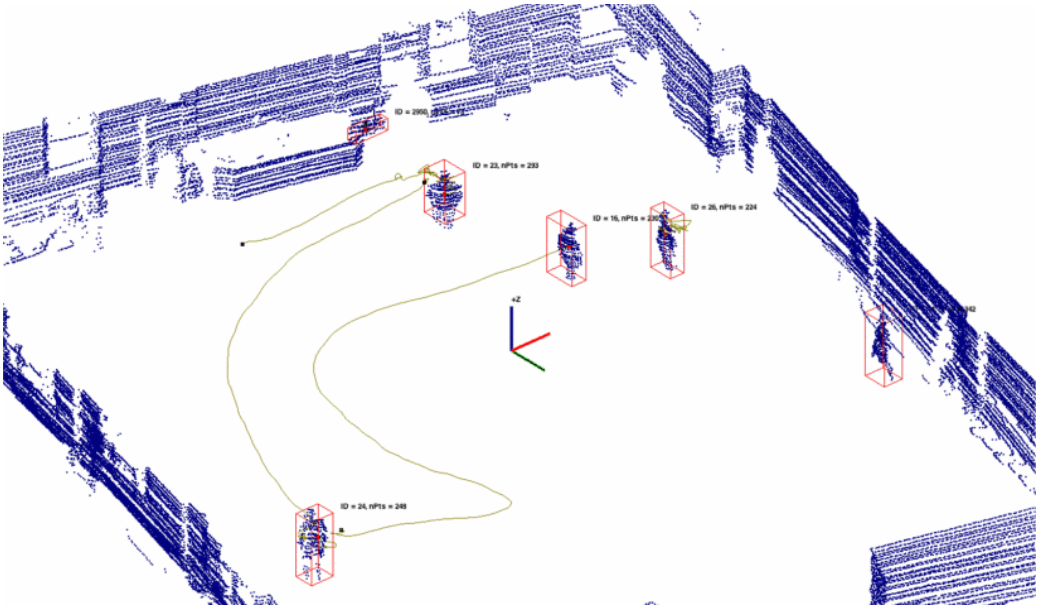


Figure 8: The model-based object detection in single-frame data (Shackleton et al., 2010).

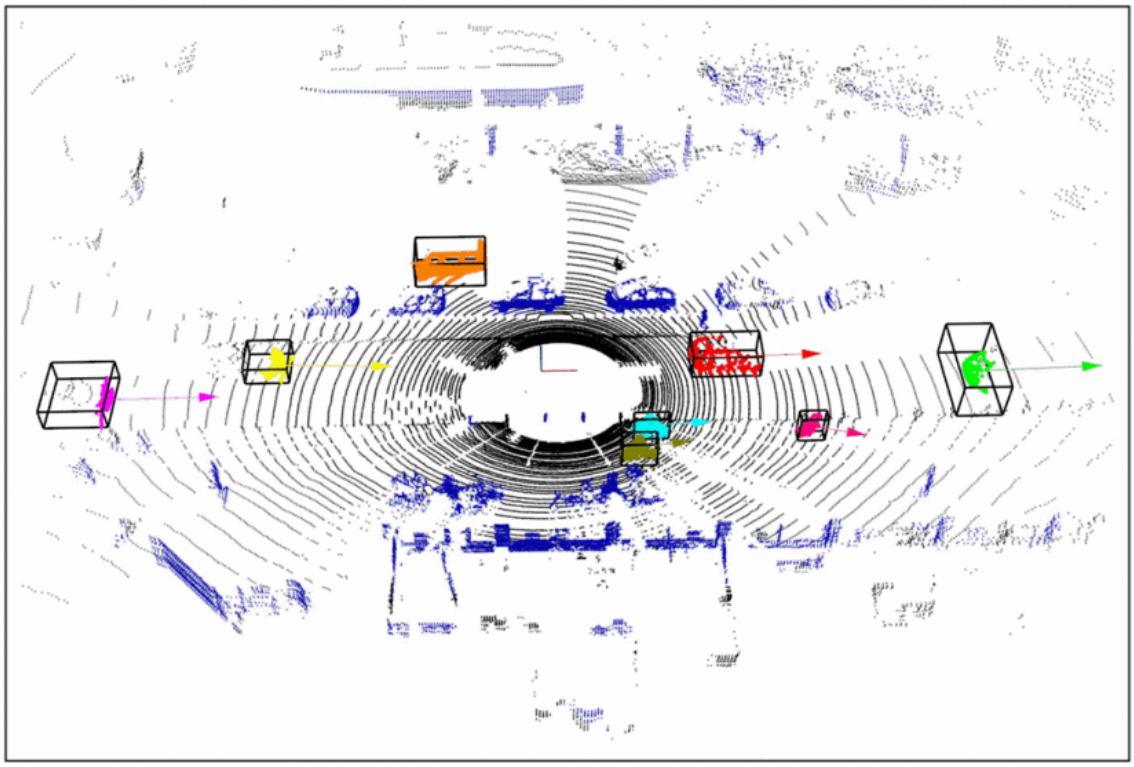


Figure 9: Objects segmented with motion cues (Dewan et al., 2016).

However, these model-based dynamic object detection and tracking methods have poor generalization ability, so they are not suitable for cases where the types of detected objects cannot be predicted completely. To avoid the limitations of prior models, some model-free methods are also proposed. For example, Dewan et al. (2016) first used Random Sample

Consensus (RANSAC) (Fischler & Bolles, 1981) to estimate the motion model and then segmented the point cloud directly with the motion cues. Figure 9 shows objects segmented with motion cues. Compared to the model-based method (Figure 8), its segmentation results contain a variety of objects.

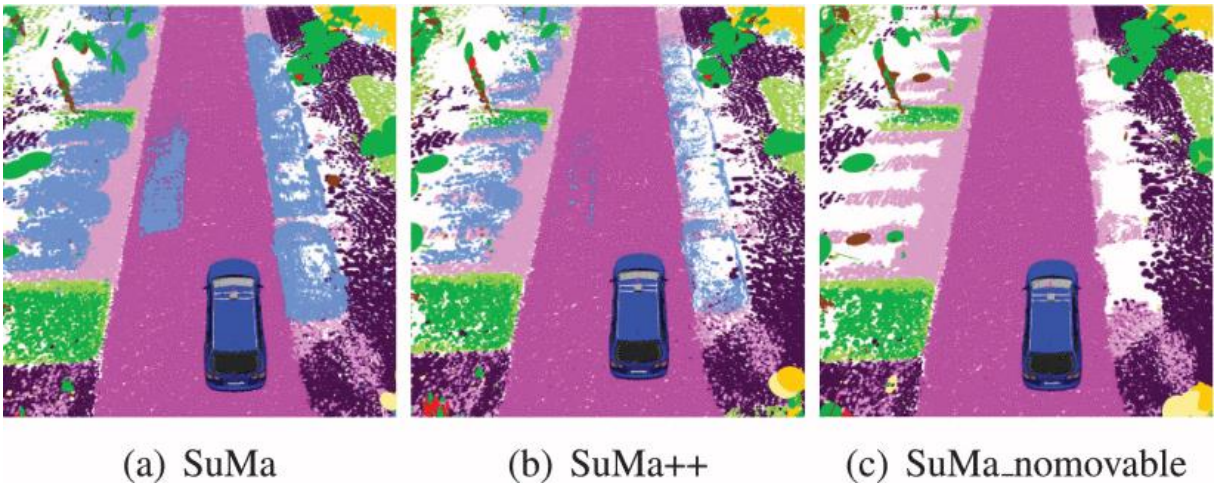


Figure 10: The dynamic object removal in SuMa++ (X. Chen et al., 2019).

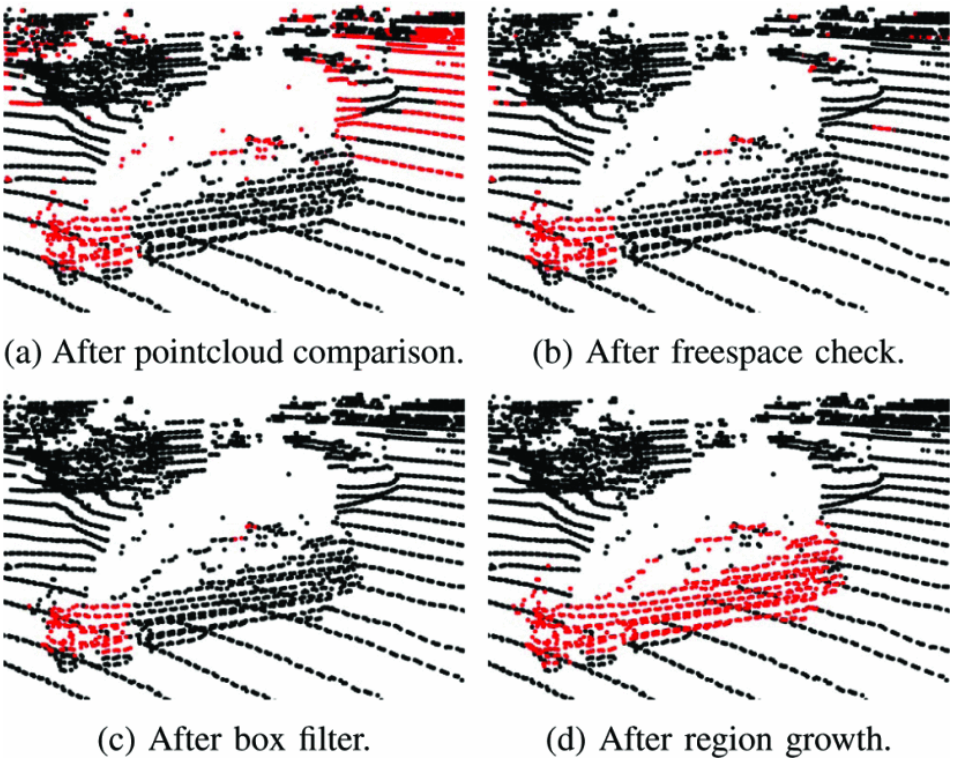


Figure 11: The main steps of the mapless and modeless online dynamic object detection method designed by Yoon et al. (2019).

As recently the semantic classification has been achieved point by point in point cloud data, some latest studies have proposed point-based dynamic detection. Therefore, dynamic tracing does not need to take the object as the basic unit but can be further achieved at the point level. X. Chen et al. (2019) proposed a Simultaneous Localization and Mapping (SLAM) method, SuMa++, which first used the Fully Convolutional Neural Network (FCN) (Milioto et al., 2019) to provide semantic class labels for each point and then detect and remove dynamic objects with the spatial semantic inconsistency (Figure 10). But this method also removes some of the static objects (see incomplete static vehicles on both sides of the road in Figure 10. (b)). Yoon et al. (2019) designed a point-based online dynamic object detection method that does not rely on prior maps or models. The main steps are shown in Figure 11. This method first uses the error metrics, which is a common concept in point cloud alignment operations, to first compare two point clouds and then extract the potential dynamic points from the unaligned points (Figure 11. (a)). Then it checks the free space (the space not permanently occupied by static points in the point cloud) by considering the spatial relationship between the LiDAR scanning ray and the surface plane that the scanned point lies on. The fake dynamic points are labeled as static points again. This step is a simplified version of the occupancy voxel grid (Figure 11. (b)). After that, a box filter (Figure 12) is used to further move outliers in the potential dynamic points (Figure 11. (c)). Finally, the remaining points will be considered as the seed points of the region growth algorithm (Moosmann et al., 2009) to extract the complete dynamic object (Figure 11. (d)).

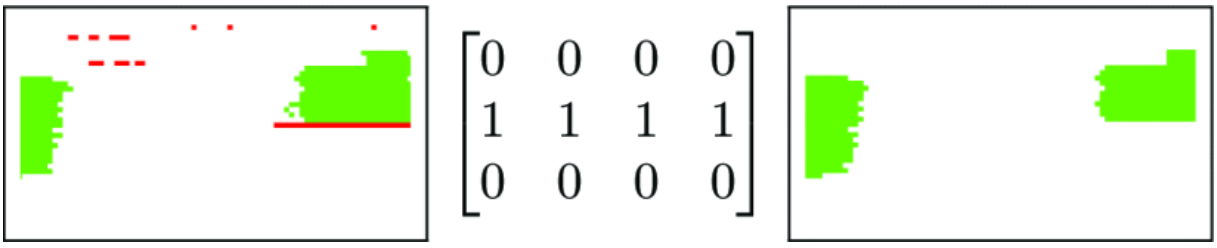


Figure 12: The box filter used to remove the outliers in potential dynamic points (Yoon et al., 2019).

Compared with object-based methods, point-based methods are more susceptible to viewpoint occlusions and data sparsity. For the method of ray tracing, another issue is that the influence of noise on free space detection is not considered, which results in labeling many static points as dynamic by mistake. Therefore, some studies use Octomap to identify dynamic objects (Arora et al., 2021; Lim et al., 2021; Pagad et al., 2020; Schauer & Nüchter, 2018; Ushani et al., 2017). It extracts dynamic objects by exploiting the spatial conflict of LiDAR rays between multi-frame data. The theoretical background of Octomap will be introduced in Section 3.1. The advantage of Octomap over other multi-frame methods is that

instead of simply marking points as static or dynamic, Octomap describes the probability of each voxel being occupied, which considers the effect of noise on ray tracing. This allows Octomap to obtain more precise free space in the point cloud.

The main problem with Octomap is that it is not a computation-friendly and memory-friendly method because Octomap introduces additional 3D grid so that the orientation, resolution, and spatial domain of the grid all affect the performance. MLS will generate point cloud data of large density and wide range. Thus MLS data results in a very large voxel grid in Octomap. Therefore, a large amount of memory is required to perform ray tracing. At present, one of the key points and difficulties of Octomap-related research is to reduce the memory burden and improve the computing speed of Octomap. One idea is to introduce visibility-based approaches (Banerjee et al., 2019; Kim & Kim, 2020). However, visibility-based approaches are severely affected by the occlusion problem.

2.3 Conclusion of Related Works

In general, although there are many well-performed static target extraction methods for single-frame data, the methods for dynamic objects are still not very mature, especially in MLS data. But some single-frame methods are instructive for multi-frame methods. For example, a single-frame method is integrated into a multi-frame method as a sub-step in many research approaches.

In addition to obtaining more points and higher point density, multi-frame data also provides a lot of additional information helpful for dynamic object extraction, such as the relative spatiotemporal relation between data frames. Among the many multi-frame methods, Octomap provides a unique point-by-point ray-tracing solution and had some good results in previous studies. But it still needs to be improved in terms of computational efficiency and memory consumption.

Compared to previous studies on Octomap-based dynamic object detection, the contribution of this thesis is to propose a method to segment the original input MLS point cloud into multiple subsets thus avoiding the generation of a huge voxel grid to achieve better efficiency and reduced memory requirements. On the other hand, the segmented MLS point cloud can more efficiently build Octomaps and extract free points by accelerating with parallel computing.

3. Theoretical Background

This chapter introduces the theoretical background of the two important concepts, the Octomap (Section 3.1) and the point-based neighborhood query (Section 3.2), which are used in the methodology of this thesis.

3.1 Octomap

Octomap is a 3D occupancy voxel grid mapping approach based on a cell's octree structure (see Figure 13, where free cells are shadowed white and occupied cells are black), which was originally developed to implement the maples and modeless 3D geometric environment representation in robotics research (Hornung et al., 2013). This mapping approach is inspired by the occupancy grid mapping proposed by Moravec and Elfes (1985). Figure 14 compares the Octomap with other common 3D representations of LiDAR data.

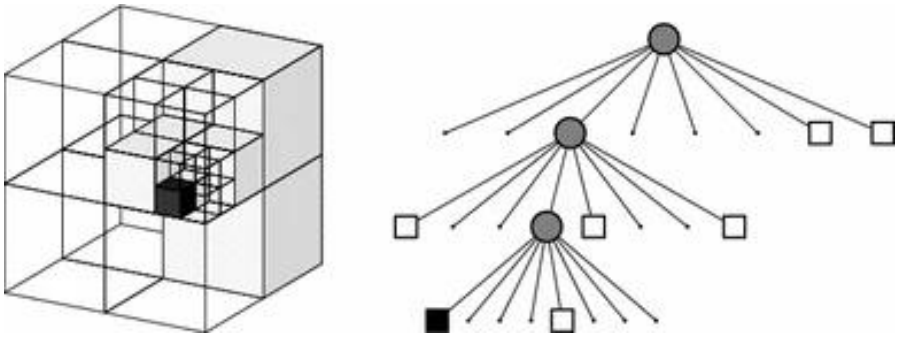


Figure 13: The volumetric model (left) and its corresponding octree representation (right) (Hornung et al., 2013).

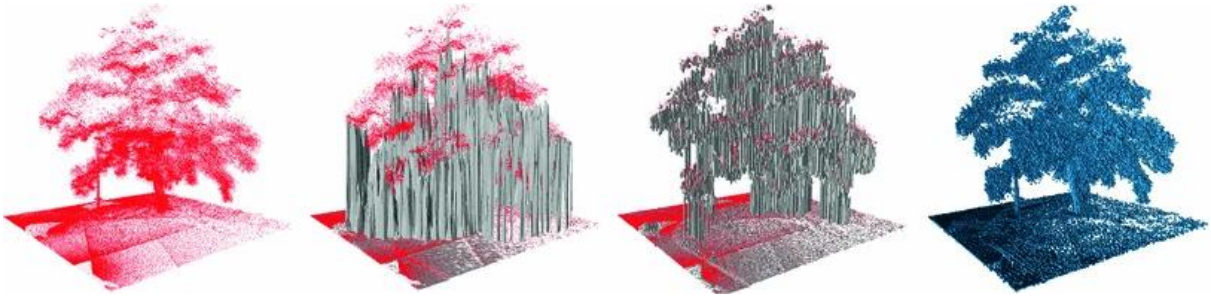


Figure 14: 3D representations of a tree scanned with a laser range sensor (from left to right): Point cloud, elevation map, multi-level surface map, and Octomap (Hornung et al., 2013).

Octomap obtains the full representation of space, which means the space is divided into three classes: unscanned space, free space, and occupied space. The free space (navigable space) and the occupied space (obstacles) are important for safe robot navigation. In addition, information about unscanned space is also coded implicitly in the map because it is also critical for some tasks such as autonomous robot exploration of an unknown environment. For ease of understanding, Figure 15. (a) uses a 2D grid to demonstrate Octomap's spatial division logic: First, all voxel cells in the initial Octomap are labeled as unscanned space by default. When the MLS sensor moves to position s_1, it emits the first ray s_1p_1 and captures the target point p_1. The Lidar ray is voxelized in Octomap using Bresenham's line algorithm (Bresenham, 1965). The voxel cell v_{p_1} where the target point p_1 is located is considered occupied, while the voxel cell v_{s_1} where the MLS sensor $s_1$is located and all the voxel cells between v_{p_1} and v_{s_1} are free.

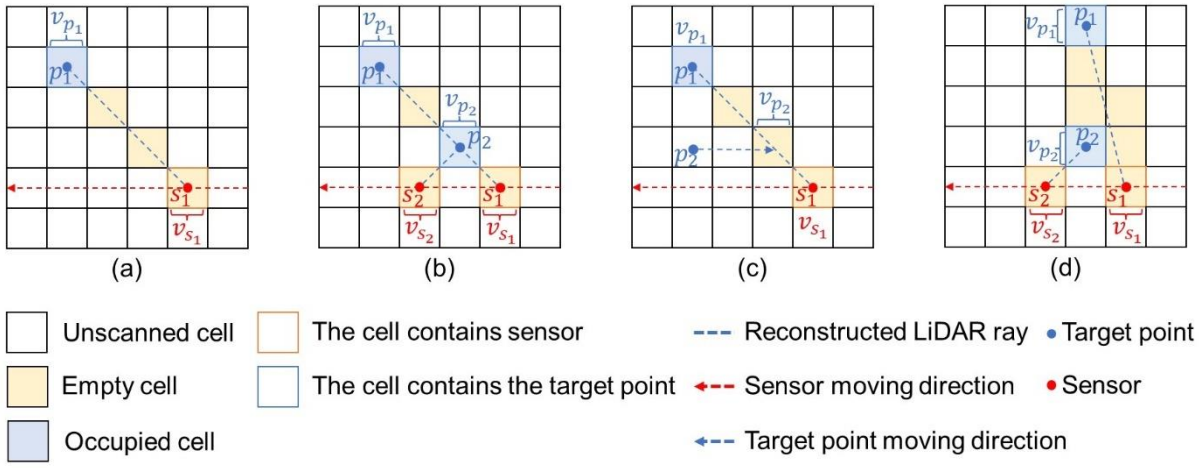


Figure 15: Spatial classification based on ray tracing and its possible spatial conflicts.

As shown in Figure 15. (a), when only one ray is inserted into the Octomap, all voxel cells are explicitly classified as unscanned, occupied, or free space. However, the situation becomes much more complicated when multiple rays are inserted. Some rays may have space conflicts with each other. Figure 15. (b) shows an example of a space conflict: After emitting ray s_1p_1, the MLS sensor continues to move in the direction s_1s_2. When the sensor reaches position s_2, it emits the second ray s_2p_2 and captures the target point p_2. Since p_2 is a point on the ray s_1p_1, there is a contradiction between s_1p_1 and s_2p_2. For s_1p_1, the voxel cell v_{p_2} , which contains the target point p_2, is free. But for s_2p_2, v_{p_2} is occupied. There are three possible reasons for this inconsistency in voxel occupancy. The first is that p_2 is a dynamic point, so when the sensor emits the first ray s_1p_1, p_2 is moving towards but not yet at v_{p_2}. So, the ray s_1p_1 is not occluded by p_2. Based on this assumption, the actual case of v_{p_2} should be free (Figure 15. (c)). The second possibility is that the conflict is caused by a

measurement error in LiDAR. p_1 in Figure 15. (b) may be the noise generated during the measurement, so its position is not correct. An assumption is made that the real position of p_1 in the actual environment is shown in Figure 15. (d). Based on this assumption, we draw an opposite conclusion: in the real world, v_{p_2} is occupied. The last possibility is that the object being scanned is a non-rigid static object or has a sparse structure. Typical examples are tree crowns and grasslands. In Figure 15. (b), if p_2 comes from a non-rigid object, then it is possible for voxels v_{p_1} and v_{p_1} to be occupied space at the same time. In a real MLS data set, the above three conditions may exist simultaneously. This makes it difficult to analyze what causes spatial conflict cell by cell. Therefore, it is impossible to directly classify voxel cells with spatial conflict without knowing the true environment.

To avoid the classification uncertainty caused by space conflicts of rays, Octomap does not directly mark the occupation status of each voxel cell but calculated the occupation probability. For each voxel cell, find all the rays that intersect it in space. If one ray is reflected within the voxel cell, this voxel cell is observed to be occupied once. If one ray traverses the voxel cell, this voxel cell is observed to be free once. The occupancy probability is obtained by calculating the ratio of the number of times the voxel cell is occupied to the total number of observations. This is very similar to the hit-and-miss approach proposed by Kelly et al. (2006).

Octomap also provides a convenient way to update probabilities. From the insertion of the second ray, the latest occupancy probability of the voxel cell is derived from the previous probability, rather than having to traverse all the inserted rays each time to obtain statistical information. For the voxel cell n, given t times sensor measurements $z_{1:t}$, its occupancy probability $P(n|z_{1:t})$ is obtained in Eq.3. In this update formula (Eq.3), z_t means the current measurement. $P(n)$ is the prior probability and $P(n|z_{1:t-1})$ is the previous estimate. $P(n|z_t)$ is the probability of voxel n to be occupied given the measurement z_t.

$$P(n|z_{1:t}) = \left[1 + \frac{1-P(n|z_t)}{P(n|z_t)} \cdot \frac{1-P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \cdot \frac{1-P(n)}{P(n)}\right]^{-1} \quad (3)$$

After determining the final occupancy probability of each voxel cell, a threshold value $thres_{occupied}$ is set to label the voxel cells with a probability greater than the threshold as occupied space, and the other scanned as free space. Unscanned cells have no probabilities and are not explicitly added to the octree structure to reduce computation and memory consumption. The 3D environment representation obtained in this way takes full account of the effects of dynamic objects, non-rigid static objects or objects with a sparse structure, and

measured noise. It is also the theoretical basis for extracting free points in Section 4.1 of this thesis.

3.2 Point-based Neighborhood Query

In several sub-steps of the methodology proposed in this paper, such as removing noise, removing vegetation, and finally extracting dynamic objects, point-based local neighborhood queries are required.

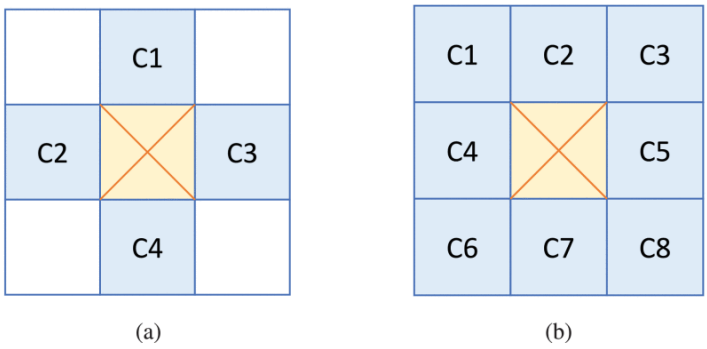


Figure 16: 4-connectivities (a) and 8-connectivities (b) of the raster data (Kampffmeyer et al., 2019).

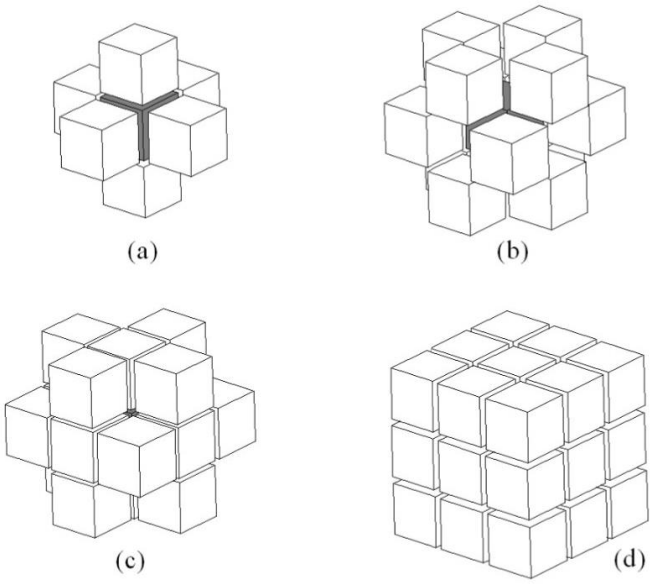


Figure 17: 6-connectivities (a), 12-connectivities (b), 18-connectivities, and 26-connectivities of the voxel data (Sánchez-Cruz et al., 2013).

For raster data, its neighborhood is usually defined by connectivity (Gonzalez & Woods, 2008). Figure 16 illustrates the two main types of connectivity for raster data: 4-connectivities which only focus on the vertical and horizontal directions and 8-connectivities which focus on horizontal, vertical, and diagonal directions. The neighborhood of a voxel object can also be similarly defined by connectivity. Figure 17 illustrates several major connectivity modes of voxel objects. However, point cloud data is discrete, so the adjacency relation between points is implicit, which means that the neighborhood definition based on connectivity cannot be directly applied to the point cloud. Therefore, it is necessary to specially define the neighborhood of point cloud data. Otepka et al. (2013) proposed that for point p_i from the point cloud, all points with a distance to p_i less than a certain threshold or the nearest k points of p_i are regarded as its neighborhood. There are two points to note in this definition. First, p_i does not need to be a real point element in the point cloud, in other words, it can be a virtual point such as the geometric center of the point cloud. Second, there is no restriction on the type of distance in the definition, so it could be 3D distance, xy-plane distance, z-axis distance, or Manhattan distance. Some researchers defined the point cloud neighborhood based on triangulation or tetrahedralization (Gorte, 2002; Maas & Vosselman, 1999). However, these two methods require additional processing steps, such as constructing Triangular Irregular Networks (TIN) or Tetrahedral Networks (TEN), which cannot be directly applied to the original point cloud.

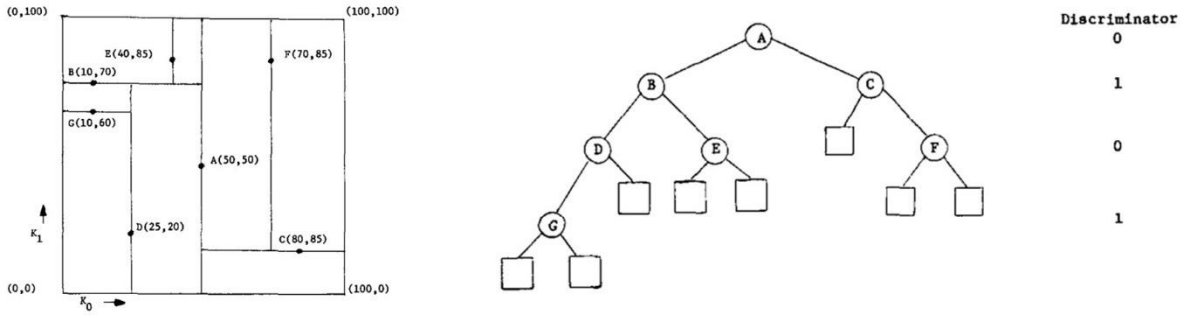


Figure 18: An example of 2D points (left) with their corresponding KD-tree structure (right) (Bentley, 1975).

The point cloud neighborhood definition proposed by Otepka et al. (2013) corresponds to two commonly used point-based local neighborhood query methods: fixed radius search and KNN search. These two methods can be efficiently implemented based on KD-tree (Bentley, 1975) structure (Figure 18). But in practice, many people are unaware of the difference in effect between these two neighborhood query methods. The wrong choice of neighborhood query method may affect the performance of many point cloud processing tasks. Otepka et

al. (2021) analyzed this problem with a generic spatial search framework: When the point distribution in the point cloud is isotropic and homogeneous, there is no obvious difference between the two neighborhood query methods. However, in the actual point cloud data, due to the influence of scan mechanics (Figure 19) and other factors, the point distribution is usually anisotropic and inhomogeneous. For example, areas far from the center of the scan usually have a lower density of captured points. The main advantage of fixed radius search is that it provides a symmetrical neighborhood, but KNN search does not. A symmetrical neighborhood means that all points are mutual neighbors under the neighborhood. Figure 20 illustrates examples of the symmetrical neighborhood from a fixed radius search (Figure 20. (a)) and the non-symmetrical neighborhood from a KNN search (Figure 20. (b)). In Figure 20. (a), the blue point and all orange points are mutual neighbors. In Figure 20. (b), the orange point is a neighbor of the blue point, but the blue point is not a neighbor of the orange point. For some point cloud processing tasks, such as performing region growth algorithms or calculating local point densities, symmetric neighborhoods are explicitly specified. But points in the symmetric neighborhood may be non-symmetrical. Although KNN search provides a non-symmetrical neighborhood, it has better neighborhood search ability in the region with density change, such as the boundary region far from the sensor in MLS data (Pfeifer et al., 2021).

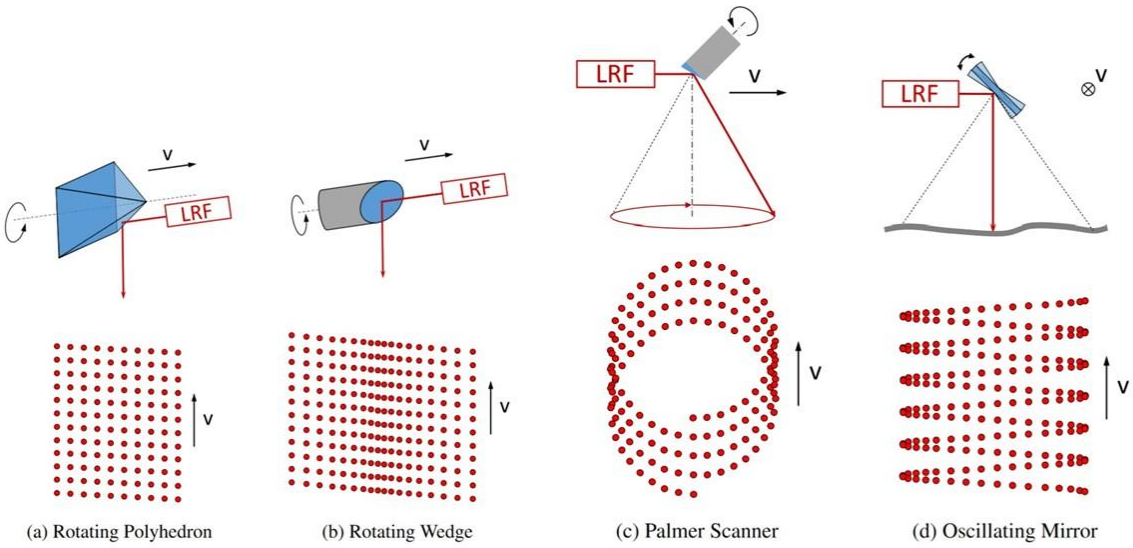


Figure 19: Some scan mechanics and their scan pattern on a horizontal planar surface (Otepka et al., 2021).

Based on the above analysis, this research takes point-based space search as the main means to query the point cloud neighborhood and selects different search strategies in

different sub-steps according to the task objectives, to achieve better performance of the proposed method.

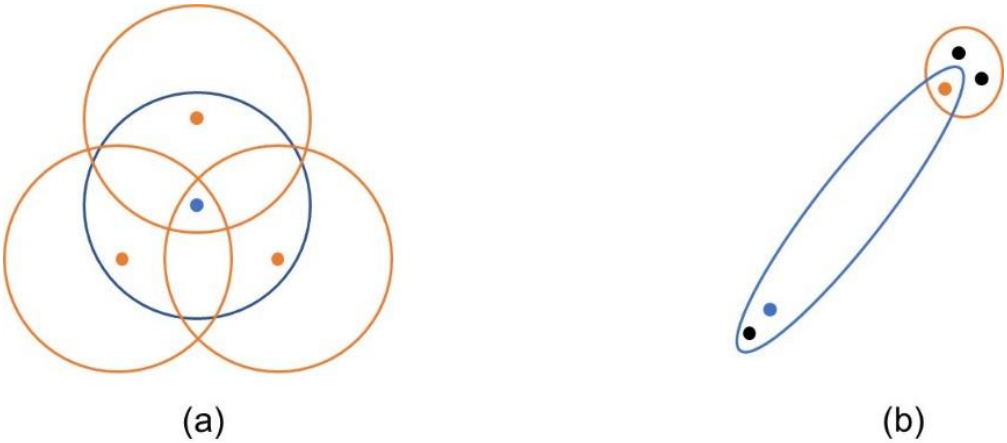


Figure 20: The symmetrical neighborhood from a fixed radius search (a) and the non-symmetrical neighborhood from a KNN search (b).

4. Methodology

This chapter discusses the methodology of this research. Implementation details of this method, such as typical values for all involved parameters, are given in Chapter 5. The main task of this proposed method is to use the MLS data and its corresponding sensor trajectory to move the dynamic objects from the original input MLS point clouds and only keep the static objects. As shown in Figure 21, the workflow is divided into five sub-steps:

(1) Extract the free points from the input MLS point cloud using Octomap (Section 4.1).
(2) Delimtate the ROI by removing the ground surface and the high-altitude space (Section 4.2).
(3) Remove noise with free-point rate from free points (Section 4.3).
(4) Remove the vegetation areas from free points using the number of returned LiDAR rays (Section 4.4).
(5) Use the filtered free points as seed points to extract the dynamic objects (Section 4.5).

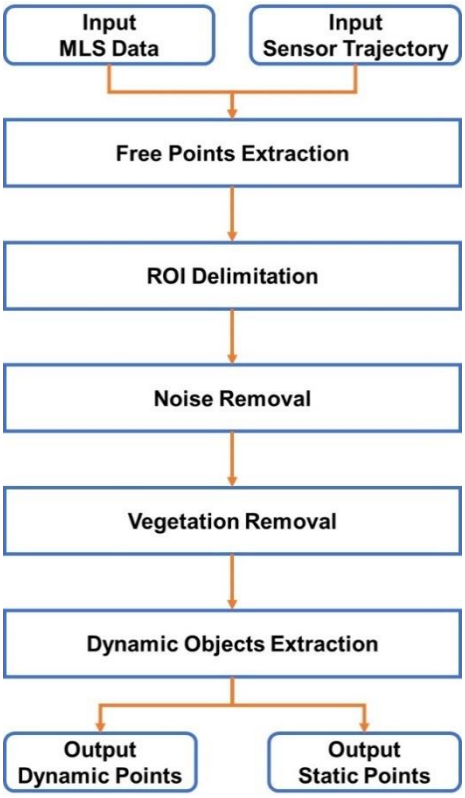


Figure 21: The main workflow of this method.

4.1 Free Point Extraction

This section describes how to extract free points from Octomap in a more efficient Octomap manner then the original Octomap method using MLS data and its corresponding sensor trajectory data. Free points here are defined as all points located in the Octomap space whose occupancy probability is less than a given threshold $thres_{occupied}$. As shown in Figure 22, the workflow of this step begins by splitting the entire MLS data into multiple data frames. Each data frame and its neighbor data frames are merged into one group and inserted into an independent Octomap. Free points are extracted from each Octomap based on an occupancy probability threshold and then merged. Finally, the expected free points are obtained after removing the redundant points. The rest of this section will introduce these operations in detail.

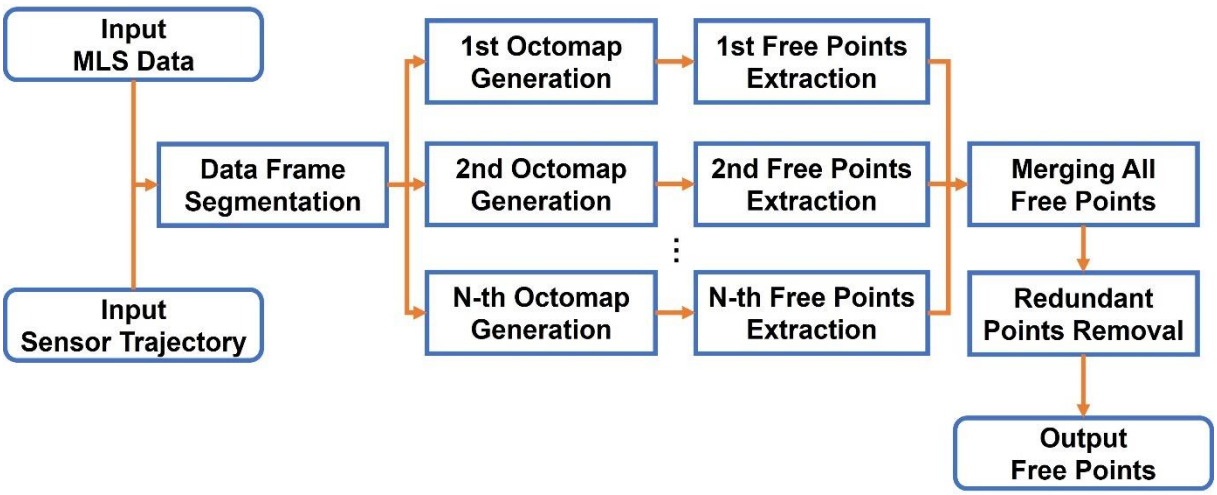


Figure 22: The workflow of free points extraction.

Octomap is generally not considered computation-friendly and memory-friendly. This problem is more prominent when dealing with high point density MLS data, especially when Octomap is set to a very small voxel size. Therefore, before generating Octomap and performing ray tracing, it is necessary to consider how to make this operation as efficient as possible. The solution used here is to split the entire MLS data into multiple sections (data frames). This avoids generating a very large voxel grid but instead generates multiple relatively small voxel grids and excludes rays emitted from very far away in each small voxel grid, thus reducing the computational effort and memory requirements. Figure 23 uses an example to further explain this idea: Octomap is built based on ray tracing. For the target section (see Figure 23. (b) and. the green box area in Figure 23. (a)), most of its intersected rays are emitted when the MLS sensor is located between points p_1 and p_2. When the MLS sensor is located at

other positions, some rays also reach the target section, but in very small quantities. Moreover, these rays have lower measurement accuracy because they are emitted by the sensor at a distance far from the target section (Pfeifer et al., 2021). Thus, when the Octomap is used to represent the target section shown in Figure 23. (b), the data collected by the sensor from positions p_2 to p_1 provides most of the highly accurate rays associated with this target area.

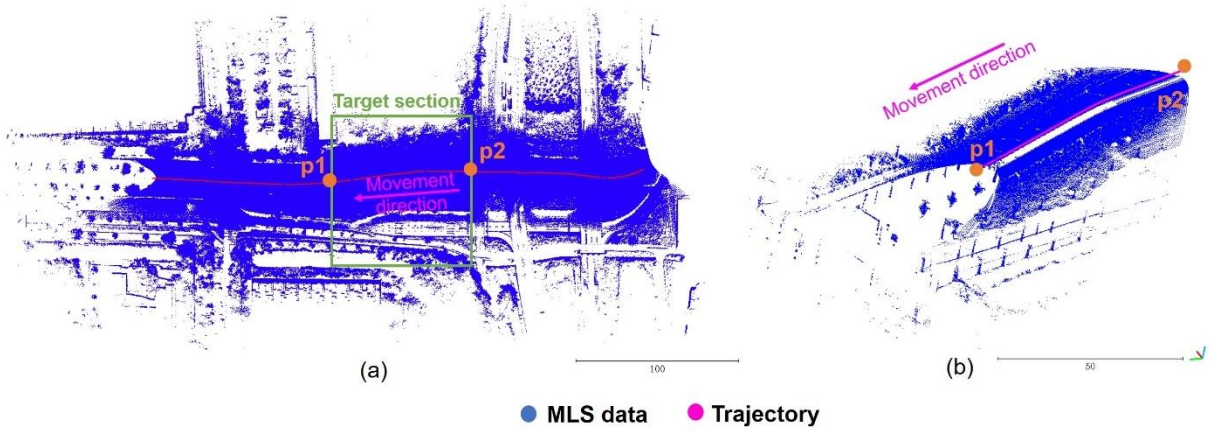


Figure 23: A section (b) from the whole MLS data (a).

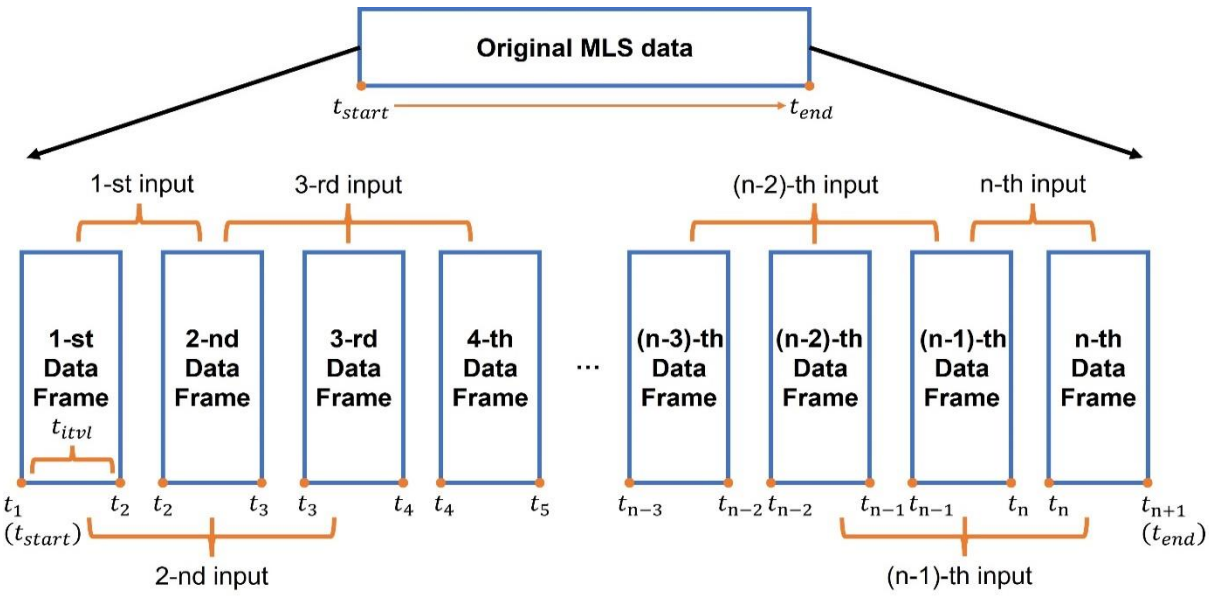


Figure 24: Data frames segmentation.

The detailed flow of data frames segmentation is given in Figure 24: Given a continuous scan of MLS data from moment t_{start} to moment t_{end}, and a time interval t_{itvl}. The original MLS

data is split into n data frames ($n = \lceil (t_{end} - t_{start}) \,/\, t_{itvl} \rceil$), each with a scan time of t_{itvl} (the scan time of the last frame may be less than t_{itvl}).

The next step is to specify the Octomap voxel size by setting the parameter $size_{voxel}$ and then build Octomaps using the segmented data frames. If each data frame is inserted directly into a separate Octomap, a problem arises: the start and end moments of each data frame (e.g., moment t_2 and moment t_3 for the second data frame in Figure 24) are missing a lot of data compared to the other scan moments because the start moment lacks preceding scan data and the end moment lacks subsequent scan data. These missing data are split into their neighbor data frames. Thus a relatively poor reconstruction result is obtained at the start and end moments of each data frame, due to the relative lack of scan data. To cope with this problem, the solution given here is to pack each data frame with its neighbor data frames and then insert them together into an Octomap, so that the missing data at the start and end moments of each data frame is filled with its neighbor data frames. Thus each Octomap build is fed three consecutive frames of data (the first and last Octomap are only fed two consecutive frames of data).

The principle and method of Octomap construction have been described in detail in Section 3.1, so this section will skip this part and discuss the free point extraction steps after Octomap construction is completed. By setting an occupancy probability threshold $thres_{occupied}$, the scanned space in each Octomap is classified into two discrete states, occupied and free (see Eq.4). All points from the two or three inserted consecutive frames that lie in the free voxel space are free points. The free points from all Octomaps are extracted and combined into one data to obtain the free points of the complete scan area. Since neighbor data frames are added in each Octomap construction, this means that each data frame is inserted into at least two Octomaps. Therefore, inevitably the free points from different Octomaps are partially duplicated, which cause unnecessary computation in subsequent steps. In addition, redundant points also cause the local density of free points to be greater than the local density of the original MLS data at the corresponding location, thus resulting in problems in the noise removal phase. So, redundant points need to be removed from the combined free points in the final operation of this section. The redundant points of each point are found by applying the fixed radius spatial search with a very small radius (close to 0) in a KD-tree structure.

$$space\ occupancy\ state = \begin{cases} occupied: if\ occupancy\ probability\ \geq\ thres_{occupied} \\ free: if\ occupancy\ probability < thres_{occupied} \end{cases} \quad (4)$$

Finally, it is important to emphasize that the free points obtained here are only considered as potential dynamic points and not directly as dynamic points. As analyzed in Section 3.1, free points can be caused by non-rigid objects, objects with sparse structure, or measurement errors (noise), in addition to dynamic objects. In a real-world LiDAR data collection environment, these three influencing factors are often difficult to avoid. Therefore, only after removing all free points due to non-rigid objects, objects with sparse structure, and potential measurement noise (outliers), the remaining free points are considered as a subset of all dynamic objects. The related processing methods will be described in the subsequent part of this chapter.

4.2 ROI Delimitation

This section aims to use the sensor trajectory, sensor mounting height, and local vehicle height restriction information to extract Region of Interest (ROI) that is relevant for land-based dynamic objects to reduce the amount of calculation in subsequent steps. ROI is defined as the space between the height of the ground surface and the maximum allowable height of a large vehicle in this research (excluding the ground). The final output data is the ROI containing the potential dynamic points. The workflow of ROI delimitation is illustrated in Figure 25.

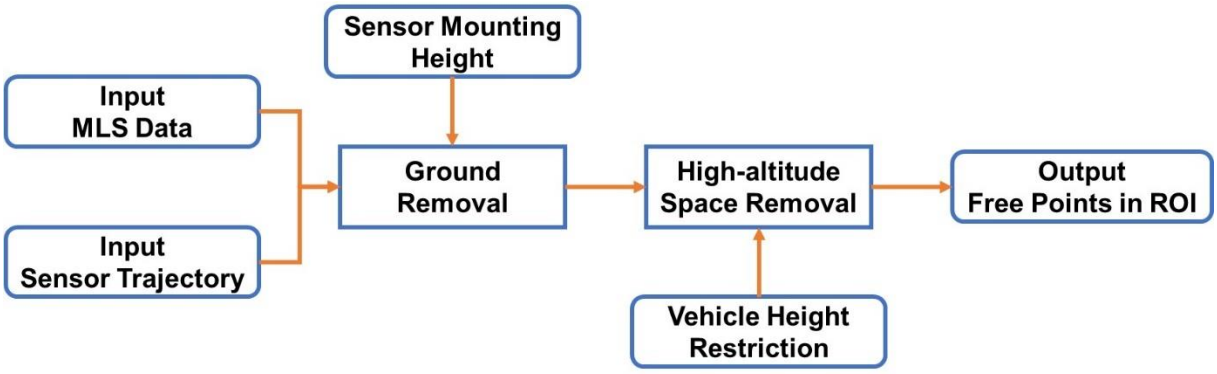


Figure 25: The workflow of ROI delimitation.

Ground removal is a common operation in dynamic object detection tasks (Choi et al., 2013; Postica et al., 2016; L. Zhang et al., 2013). This is based on two main reasons: The first reason is that the ground is usually dense and does not have any dynamic objects in the MLS data. So removing the ground reduces the computational cost of the subsequent steps without affecting the final result. Another reason is that the land-based dynamic objects are in contact with the ground, and if the ground points are not removed it is difficult to avoid

extracting some of the ground points as part of dynamic objects by accident. Thus, the integrity of the static environment might be destroyed. By removing the ground points, the objects are not connected by the ground surface, so they are more easily segmented into separate objects for detection and tracking tasks (Arora et al., 2021).

There have been many studies on ground extraction from MLS data, and most of them require a series of processing steps of the point cloud data (Che et al., 2019). However, to obtain the ground rapidly, an efficient ground extraction method based only on the sensor mounting height without additional processing of the point cloud is proposed here. Given a 3D position of MLS sensor s_i (xyz coordinates: $x_{s_i}, y_{s_i}, h_{s_i}$) from the sensor trajectory and the mounting height of the sensor h_{sm}, the height of local ground surface h_{min_i} is calculated by $h_{min_i} = h_{s_i} - h_{sm}$. If the sensor captures a point p_i (xyz coordinates: $x_{p_i}, y_{p_i}, h_{p_i}$) at position s_i and the ground height does not change within a certain range, the ground height corresponding to p_i is also h_{min_i}. So, Eq.5 determines whether the captured point p_i is a ground point by comparing h_{min_i} and h_{p_i}.

$$ground\ point = \begin{cases} true: if\ h_{min_i} \geq h_{p_i} \\ false: if\ h_{min_i} < h_{p_i} \end{cases} \quad (5)$$

While most studies related to dynamic object detection have mentioned ground removal, few studies have focused on the high-altitude space in MLS data. The high-altitude space in this context means the space above the common land-based dynamic objects, such as large vehicles. The high-altitude space in the MLS data does not include any land-based dynamic objects but may include the upper part of some street-facing buildings and the crowns of some tall trees, which cause obstacles to dynamic object detection and extraction. Therefore, removing the high-altitude space not only reduces the computational cost of the subsequent steps by decreasing the number of free points but also reduces the difficulty of dynamic object detection and extraction.

One of the difficulties in removing high-altitude space is that, unlike the obvious boundary between ground and non-ground space, the boundary between high-altitude and non-high-altitude spaces usually relies on artificial settings. Since MLS data focuses on roads and their surroundings, large vehicles are usually the largest dynamic objects in MLS data. The governing bodies of a region or country usually set the height restriction for local large vehicles in the form of regulations. Vehicles that exceed the height restriction may not be able to safely pass-through local transportation facilities such as tunnels. Therefore, the

height restriction of large vehicles is a reasonable boundary between high-altitude and non-high-altitude spaces in MLS data.

Given the height restriction of large vehicles h_{vr}, the height of boundary between high-altitude and non-high-altitude spaces at position s_i (h_{max_i}) is calculated by $h_{max_i} = h_{min_i} + h_{vr}$. Thus, Eq.6 determines whether the captured point p_i is a high-altitude point by comparing h_{max_i} and h_{p_i}.

$$high - altitude\ point = \begin{cases} false: if\ h_{max_i} > h_{p_i} \\ true: if\ h_{max_i} \leq h_{p_i} \end{cases} \tag{6}$$

Eq.5 and Eq.6 are further integrated into a single discriminant (Eq.7). Figure 26 shows this complete spatial discriminant model with three example points (p_{i-1}, p_{i-2}, and p_{i-3}). Based on Eq.7, it is known that p_{i-1}, belongs to the ground, p_{i-2} belongs to the high-altitude space, and only p_{i-3} belongs to the target space. So only p_{i-3} is kept after extracting the ROI.

$$space\ category = \begin{cases} ground: if\ h_{min_i} \geq h_{p_i} \\ target\ space: h_{min_i} < h_{p_i} < h_{max_i} \\ high - altitude\ space: if\ h_{max_i} \leq h_{p_i} \end{cases} \tag{7}$$

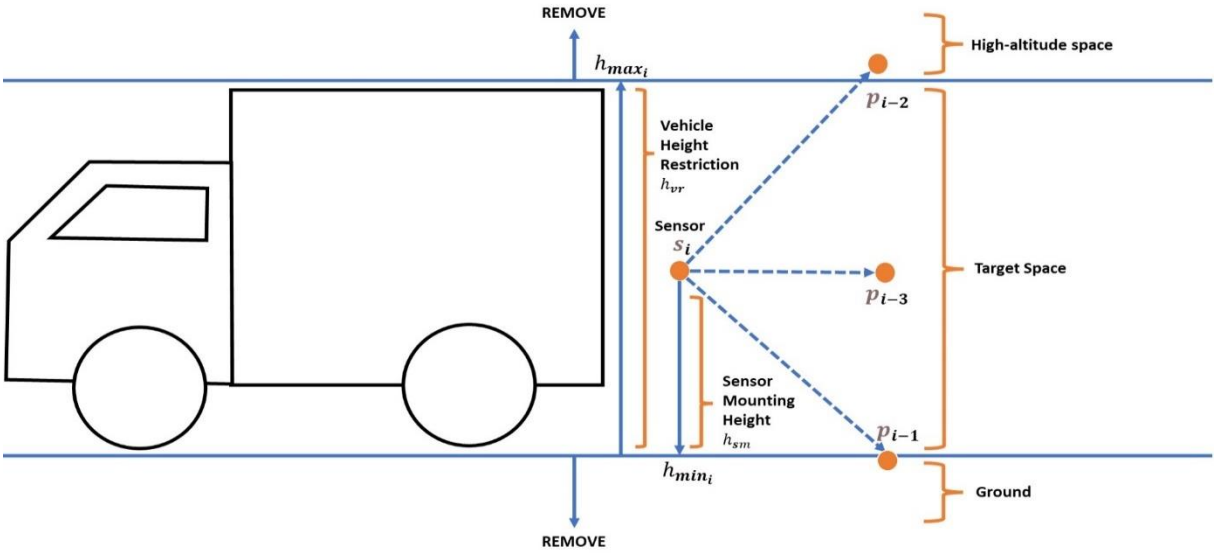


Figure 26: Spatial segmentation discriminant model.

Finally, it should be noted that the order of the ROI delimitation operation in this study is different from many other dynamic object detection studies in the whole workflow. For most related studies, ROI delimitation is performed before dynamic object detection to reduce the

detection difficulty as well as the computational cost. However, for the Octomap-based dynamic detection method, the ROI delimitation must be done after the extraction of free points. The reason is that although ground and high-altitude spaces do not contain dynamic objects, they still contain LiDAR rays that are helpful for Octomap to calculate the occupancy probability of the target space more accurately. Figure 27 shows how ground points and high-altitude points reduce the occupancy probability of the space which contains the vehicle in Octomap. Therefore, they must be retained during the construction of the Octomap stage.

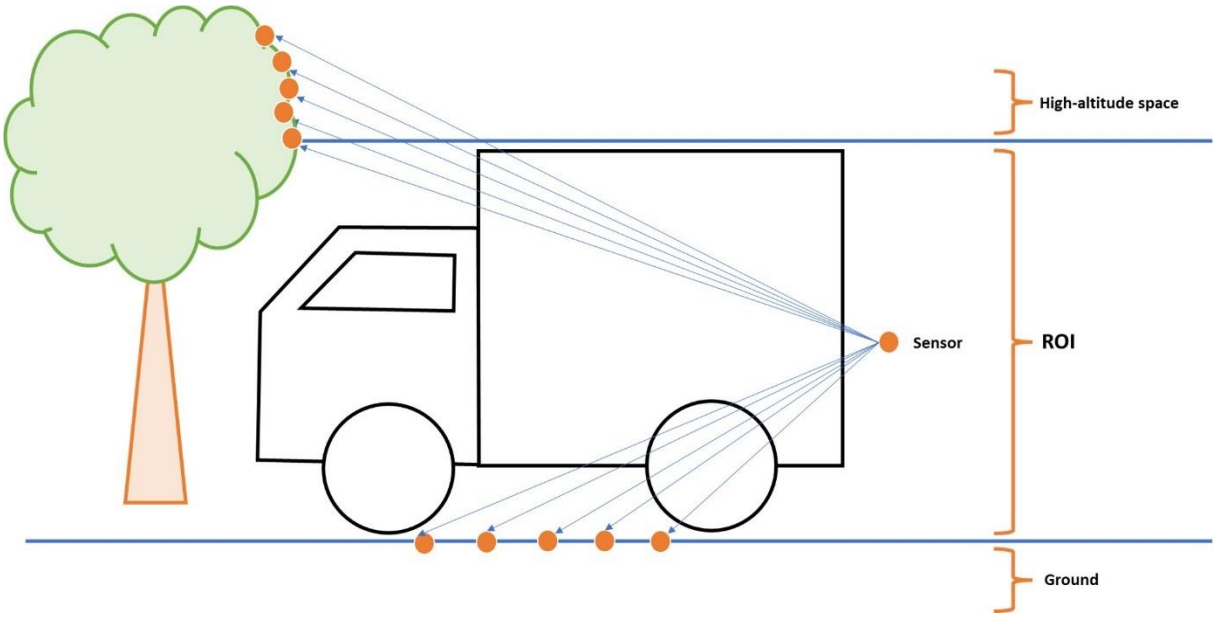


Figure 27: Ground and high-altitude space allow for a more accurate occupancy probability of the target space in Octomap.

4.3 Noise Removal

This section focuses on the removal of noise caused by measurement errors from the free points based on the free-point rate. Figure 28 illustrates the workflow of noise removal.

The measurement error of MLS is one of the potential causes of free points in Octomap. Therefore, to obtain the dynamic points from the free points, noise caused by measurement errors must be removed first. The causes of measurement errors are diverse and divided into instrument noise and environmental noise. The former includes detector noise, electronic noise, and other noise caused by the instrument itself. The latter includes optical scattering, atmospheric scattering, background light, and the different reflectivity caused by the color,

texture, and material of the object (Arisholm et al., 2018; Xu et al., 2015). However, direct denoising of the free point data based on density or fixed-radius neighborhood search is undesirable because the overall point distribution of the MLS data is anisotropic and inhomogeneous. Such methods may indistinguishably remove all dynamic points and noise in low-density areas, such as in the border regions away from the sensor in MLS data.

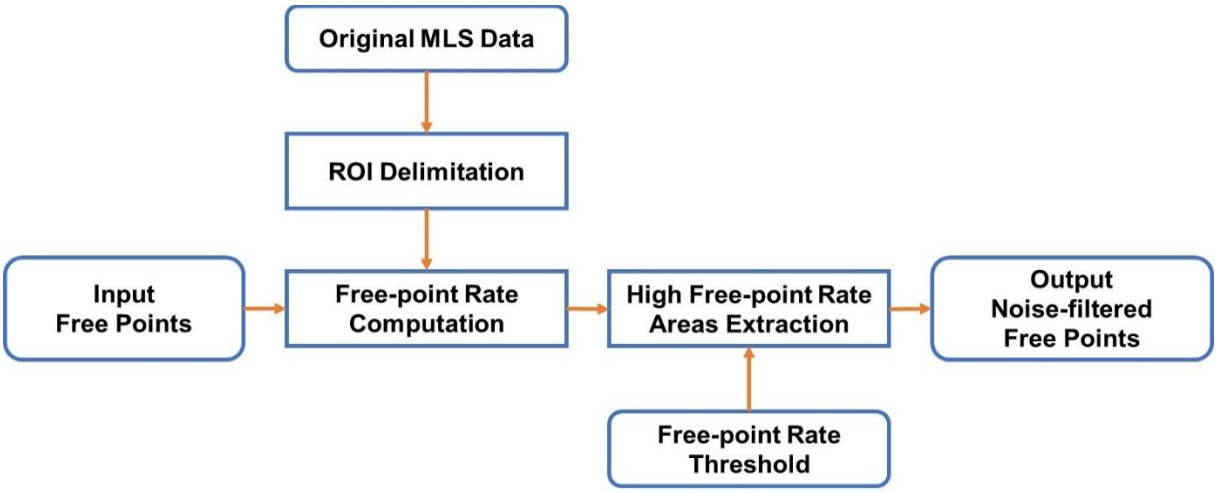


Figure 28: The workflow of noise removal.

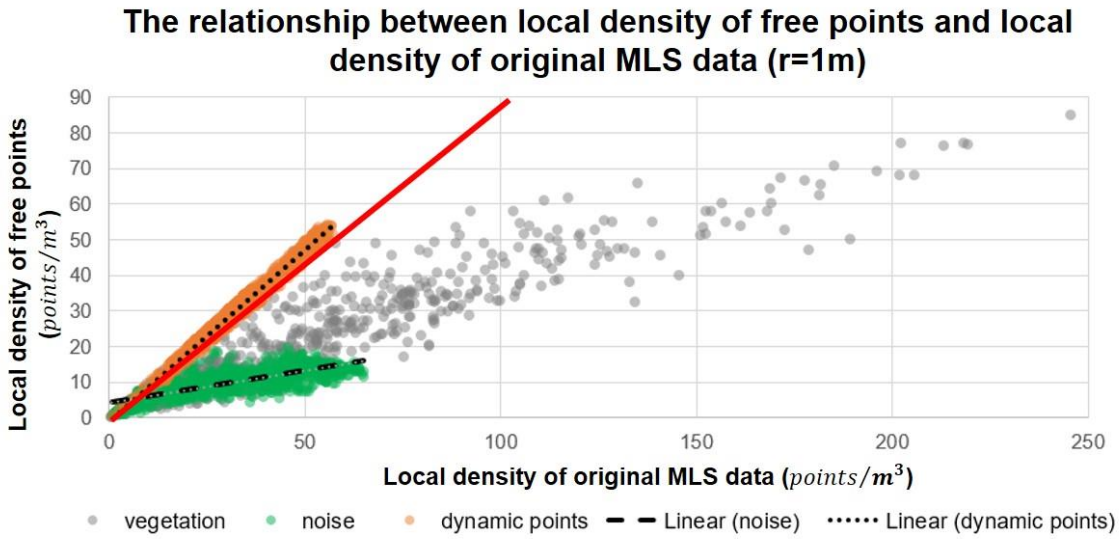


Figure 29: The relationship between the local density of free points and the local density of original MLS data (r=1m).

The noise removal method used in this section is based on a hypothesis that there is a difference between the proportions of noise and dynamic points to the object in which they

are located. This hypothesis is further explained as noise generally accounts for only a small fraction of the object in which it is located, while most of the points of a dynamic object are dynamic points. This hypothesis is evidenced in Figure 29, which shows the relationship between the local point density of 1,000 dynamic points, 1,000 noise points, and 1,000 vegetation points randomly selected from all free points in all free points and their local point density in the original MLS point cloud. These 3,000 points have been manually classified and labelled in advance. Since vegetation is not the focus of this subsection, only the noise and dynamic points are analyzed here. Both noise and dynamic points show a certain linear relationship, with the linear relationship of dynamic points being particularly obvious. The slope of the line fitted by the dynamic points is close to 1, which is significantly larger than the slope of the line fitted by the noise points. Since the slope values of the two fitted lines in Figure 29 are equivalent to the average proportion of dynamic and noisy points in their respective corresponding objects. So, this proves the hypothesis made at the beginning of this paragraph that in an ideal state the proportion of dynamic points in dynamic objects should be close to one hundred percent, much larger than the proportion of noise points in their corresponding objects. Based on the difference in slope of the two fitted lines, a free-point rate threshold $thres_{fp}$ (see red line in Figure 29, free-point rate means the ratio of the number of free points to the number of all scanned points in a certain neighborhood) is used to distinguish the noisy and non-noisy free points.

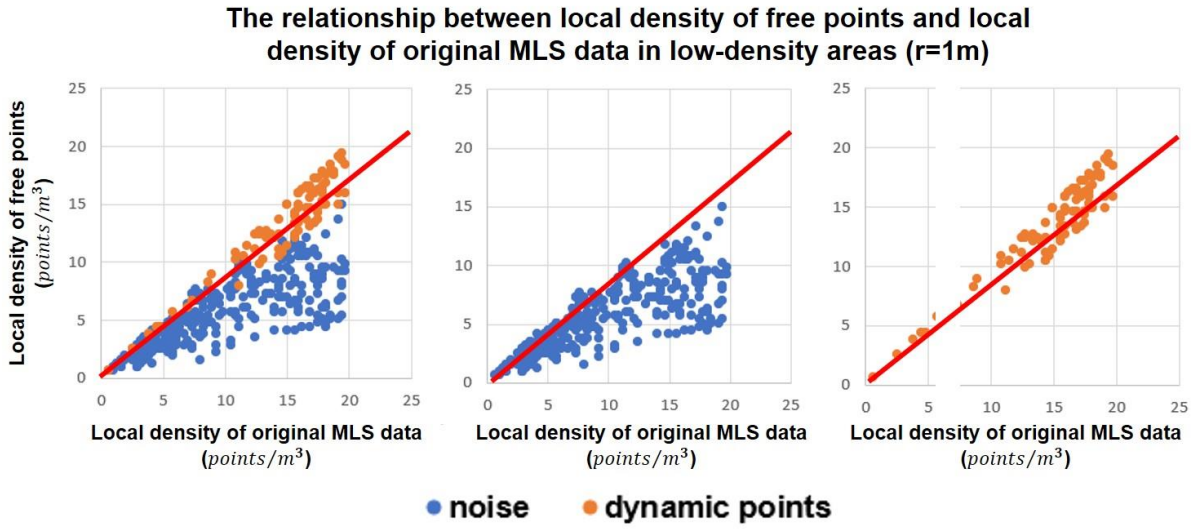


Figure 30: The relationship between the local density of free points and the local density of original MLS data in low-density areas (r=1m).

Figure 30 further illustrates the situation in the low-density areas: the noise removing method based on the threshold $thres_{fp}$ removes most of the noise in the low-density areas at the

cost of a small loss of dynamic points, if the value of $thres_{fp}$ is reasonable. Figure 31 shows the free-point rates for 1000 random dynamic points and 1000 random noise. The red line in the figure (i.e., the free-point rate threshold) splits the two curves in a good way, thus also demonstrating the feasibility of the proposed noise removal method.

The detailed process for distinguishing between noise and non-noise using the free-point rate threshold $thres_{fp}$ is as follows: First, extract the ROI from the original MLS point cloud data using the method described in Chapter 4.2, and keep only the target space containing dynamic objects. Then for each free point p_i, all its neighbor points are searched in the processed original MLS pint cloud with a radius r_{ns}, and the number of its neighbor points (num_{nb_i}) is counted. After that, the number of free points (num_{nb-fp_i}) in its neighbor points is counted. The free-point rate of point p_i ($rate_{fp_i}$) is obtained from $rate_{fp_i} = num_{nb-fp_i}/num_{nb_i}$. Given the free-point rate threshold $thres_{fp}$, the point p_i is classified as a noisy or non-noisy point based on Eq.8, by comparing $rate_{fp_i}$ and $thres_{fp}$.

$$noise = \begin{cases} true: if\ rate_{fp_i} < thres_{fp} \\ false: if\ rate_{fp_i} \geq thres_{fp} \end{cases} \tag{8}$$

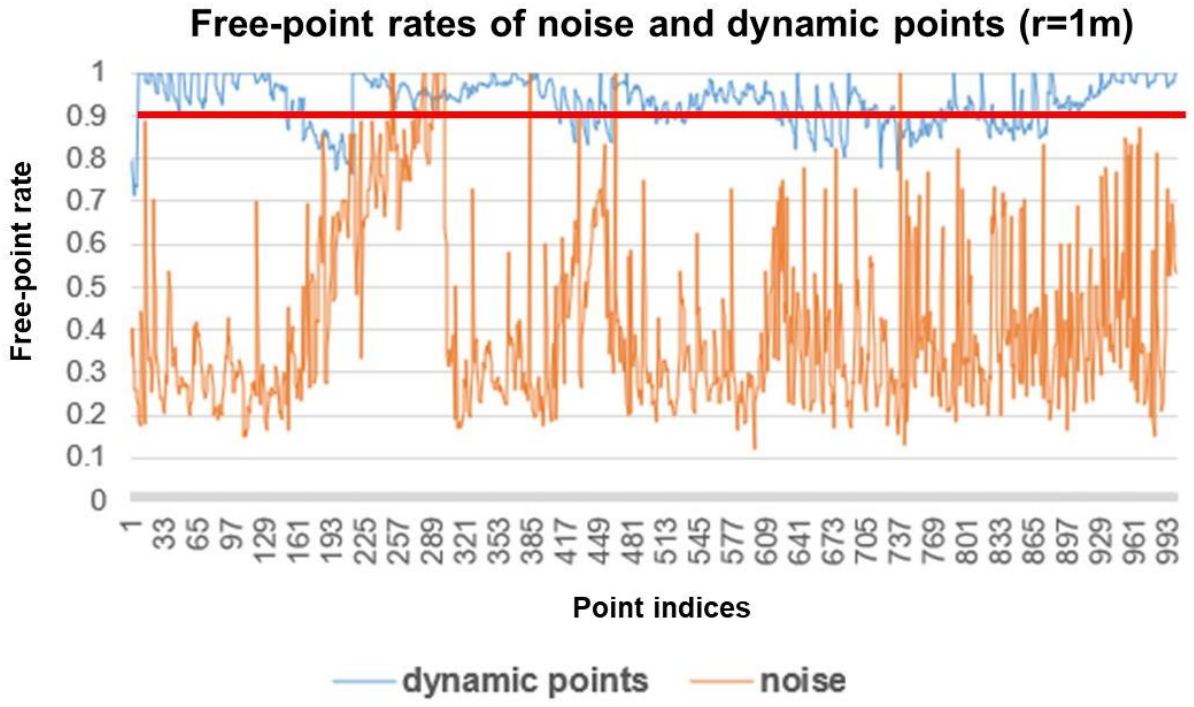


Figure 31: Free-point rates of noise and dynamic points (r=1m).

By this method, the free-point rate is calculated point by point, and then the areas with high free-point rates are extracted from the free points. These extracted free points are considered to have removed most of the noise caused by LiDAR measurement errors. Finally, it should be mentioned that the 3000 free points used in this section to demonstrate the solution are not taken from the four case sites used in Chapter 5. Therefore these 3000 points do not affect the validation of the generalization ability of the proposed noise removal method in Chapter 5.

4.4 Vegetation Removal

The purpose of this section is to remove vegetation from free points using the number of returned LiDAR rays. This is also the last processing step to obtain dynamic seed points from the free points. Figure 32 points out the workflow of the vegetation removal.

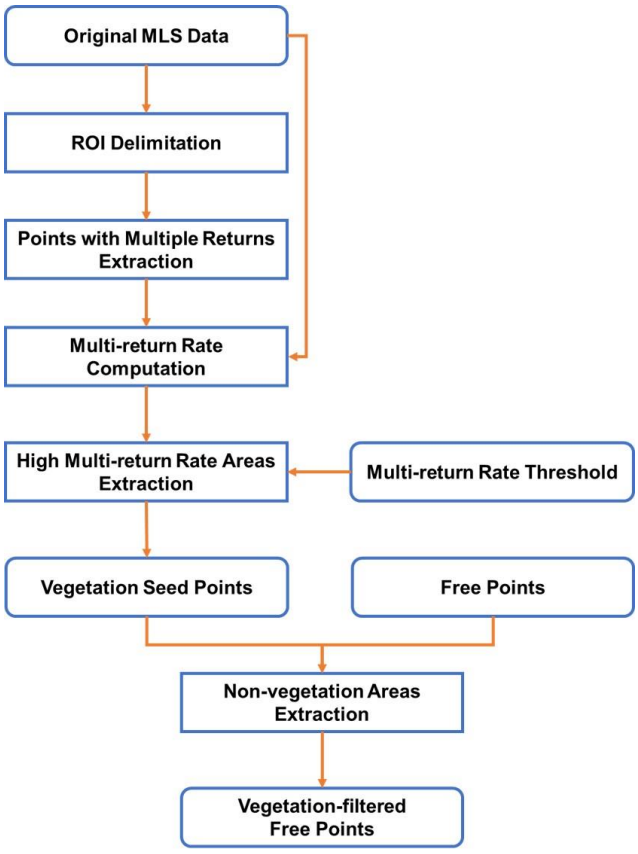


Figure 32: The workflow of vegetation removal.

As analyzed in Section 3.1, non-rigid objects and sparsely structured objects also contribute to free points, and the most common objects that fit these two characteristics in the MLS data

are vegetation. In addition, the very large dispersion in the distribution of vegetation points in Figure 29 means that the denoising results based on the free point rate threshold may inevitably include some vegetation points, because the slope values of some vegetation points in Figure 29 may be greater than the given threshold. These vegetation points interfere with the final dynamic point extraction, so they must be removed in advance.

There have been many related studies focusing on using LiDAR data for vegetation extraction in urban environments (Q. Guo et al., 2021). Several previous studies have demonstrated that the number of returned LiDAR rays and vegetation are closely related (Balado et al., 2018; Dalponte et al., 2009; Gupta et al., 2020). The reason for multiple returns from LIDAR is that when the sensor emits multiple rays, a captured object may not be able to completely block all of the LiDAR rays due to its structure or material, allowing some of the rays to pass through this object and capture other objects behind it. Thus, the sparse structure of the vegetation can easily produce multiple returns in the LiDAR data. Specifically, a laser pulse may half-hit a leaf or branch and cause multiple returns. But vegetation is not the only object that generates multiple returns. For example, glass, which is widely used in buildings and vehicles, can also generate multiple returns. This poses a challenge for extracting vegetation directly using the number of returned LiDAR rays.

Although multiple objects can produce multiple returns, the proportion of points with multiple returns varies across objects. For example, glass that generates multiple returns usually represents only a small portion of the entire building or vehicle, while for vegetation, such as tree crowns and grasses, almost all parts generate multiple returns. So, the ratio of the number of multi-returned points to the number of points in the original MLS data (i.e., the multi-return rate) is calculated in the same neighborhood using a method similar to the noise removal method in Section 4.3. Then a multi-return rate threshold $thres_{mr}$ is used to extract the vegetation seed points and used KNN search (set the number of nearest neighbors as k_{vg}) to detect and remove all vegetation points in free points.

The detailed process for distinguishing between vegetation and non-vegetation point using the multi-return rate threshold $thres_{mr}$ is as follows: First, extract all multi-returned points from the original MLS data which has been delimitated the ROI. Then for each multi-returned point p_i, all its neighbor points are searched in the processed original MLS point cloud with a radius r_{vg}, and the number of its neighbor points (num_{nb_i}) is counted. After that, the number of multi-returned points (num_{nb-mr_i}) in its neighbor points is counted. The multi-return rate of point p_i ($rate_{mr_i}$) is obtained from $rate_{mr_i} = num_{nb-mr_i}/num_{nb_i}$. Given the multi-return rate

threshold $thres_{mr}$, the point p_i is confirmed whether it is a vegetation seed point based on Eq.9, by comparing $rate_{mr_i}$ and $thres_{mr}$.

$$vegetation\ seed = \begin{cases} true: if\ rate_{mr_i} < thres_{mr} \\ false: if\ rate_{mr_i} \geq thres_{mr} \end{cases} \tag{9}$$

After getting the vegetation seed points, all vegetation points are extracted based on Algorithm 1. The final task of this section is to remove all vegetation points from free points. But the free points are sparser compared to the original MLS points and the denoising operation in Section 4.3 may further affect the neighborhood relationship of some vegetation points. Therefore, it is difficult to directly extract all vegetation points from free points using vegetation seed points. A better choice is to firstly extract the vegetation from the original MLS data compared to the free points. The first step can be performed before Section 4.1. Then the second step is to find the intersection of the extracted vegetation points and the free points and remove this intersection from the free points to obtain the non-vegetation free points. The second step needs to be performed after Section 4.3 to avoid the interference of noise in the free points.

Algorithm 1. Object Extraction with KNN Spatial Search

Input: Points to be extracted $\{P_{input}\}$, seed points $\{P_{seeds}\}$, number of nearest neighbors k

Output: Extracted points $\{P_{extracted}\}$

// Define a global point list $\{P_{checkList}\}$

$i = 0$

For p_{seed} in P_{seeds} **do**

  // Define a local point list $\{P_{object}\}$ in the for-loop

  $\{P_{checkList}\} \leftarrow p_{seed}$

  While $\{P_{checkList}\}$ is not empty **do**

    $p_{check} \leftarrow \{P_{checkList}\}.lastPt$

    $\{P_{checkList}\}.popback()$

    $\{P_{knn}\} \leftarrow$ the k nearest neighbors of p_{check} in $\{P_{input}\}$

    For $p_{neighbor}$ in P_{knn} **do**

      If $p_{neighbor}.isVisited = False$ **do**

        $p_{neighbor}.isVisited \leftarrow True$

        $\{P_{checkList}\} \leftarrow p_{neighbor}$

        $\{P_{object}\} \leftarrow p_{neighbor}$

      End if

    End for

  End while

  If $\{P_{object}\}$ is not empty **do**

For p_{obj} in P_{object} **do**

$p_{obj}.objectIndex \leftarrow i$

$\{P_{extracted}\} \leftarrow p_{obj}$

End for

$i \leftarrow \mathrm{i} + 1$

End if

End for

Return $\{P_{extracted}\}$

4.5 Dynamic Objects Extraction

This section describes the last step of the proposed method, i.e., how to extract dynamic objects from the original MLS point cloud using filtered free points as seed points with KNN spatial search. Figure 33 illustrates the workflow for extracting dynamic objects.

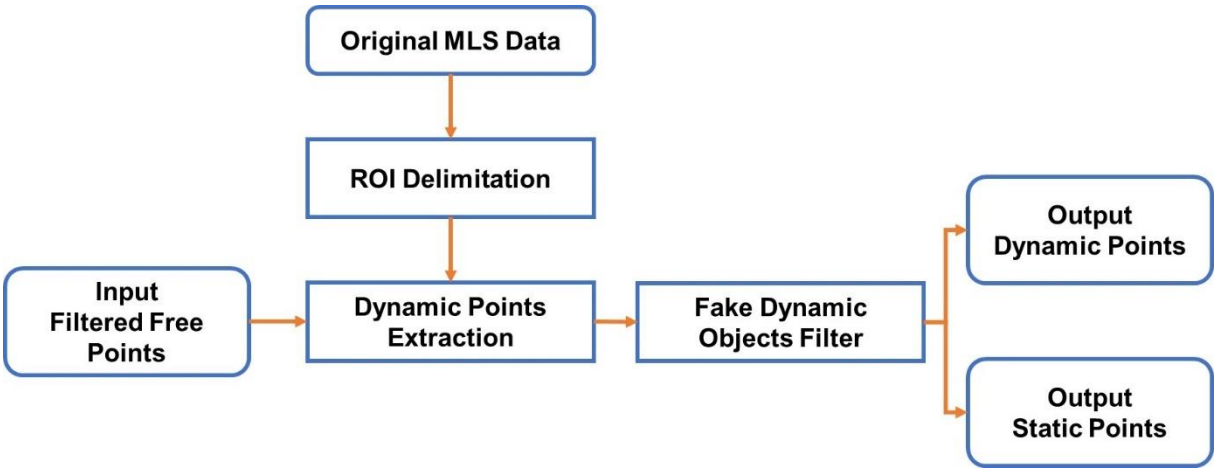


Figure 33: The workflow of dynamic objects extraction.

After the previous series of filtering processes, the remaining free points are regarded as a subset of all dynamic objects, so they are used as the seed points for extracting all dynamic objects from the original MLS data. The original MLS data needs to be advanced to extract the ROI, including ground and high-altitude areas, using the method described in Section 4.2 to better implement the dynamic object extraction. The specific KNN search-based extraction method is detailed in Algorithm 1, and the number of nearest neighbors is set to k_{do}.

Although the previous noise and vegetation filtering operations have removed most of the vegetation and noise points from the free points, a very small amount of non-dynamic points may inevitably remain in the remaining free points. These residual non-dynamic points may

lead to two kinds of fake dynamic objects: (1) returning an object that contains only a few points, or (2) returning an object with a small proportion of seed points. The reason for the first result may be that several non-dynamic free points in relatively proximity are identified as a micro-object. For such extraction results, a minimum point limit num_{min} is set for a single dynamic object. Only the extracted object whose point number is greater than the limit num_{min} is a valid dynamic object. The second type of fake dynamic object is mainly caused by a small number of non-dynamic points falling near the vegetation. These non-dynamic points act as seed points to misidentify a connected grassland or urban forest as a large dynamic object. For this type of extracted object, the seed points are only a small fraction of all points. So, the proportion of seed points (i.e., seed-point rate $rate_{sp_i}$) is computed for each extracted dynamic object candidate o_i and then a threshold of seed point proportion $thres_{sp}$ is set to filter real dynamic objects from all extracted objects (see Figure 34).

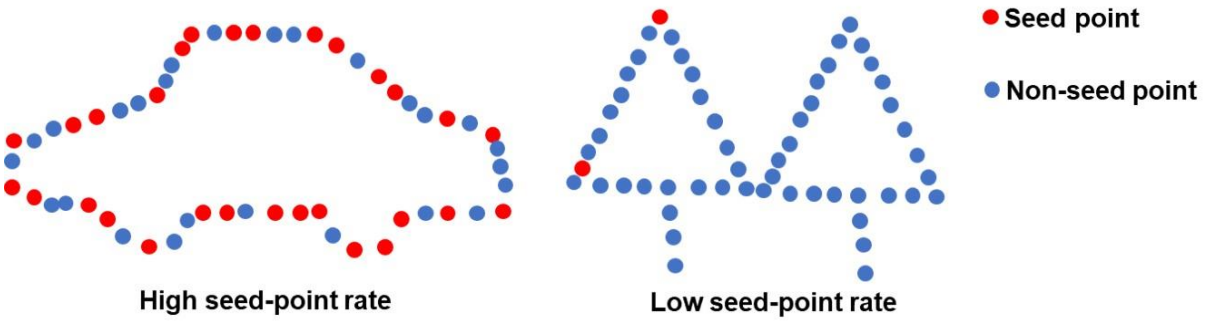


Figure 34: The objects with the high seed-point rate (left) and low seed-point rate (right).

Finally, after removing the above two types of fake dynamic objects, the real dynamic objects are obtained.

5. Implementation

This chapter is concerned with the implementation details of the method proposed in this thesis. Firstly, Section 5.1 presents the dataset used in this study and introduces the information about the four case sites. Then Section 5.2 lists the values of the relevant parameters set in this implementation. Finally, Section 5.3 presents the relevant tools used in this research.

5.1 Dataset

Figure 35: Cyclomedia's mobile data acquisition vehicle (source: Cyclomedia).

The MLS data and corresponding sensor trajectories used in this research were collected by CycloMedia[1], a Dutch environmental visualization data provider, in Delft, the Netherlands, in July 2021. The LiDAR sensor used to acquire this dataset is Velodyne's HDL-32E[2], which has an accuracy of ±2 cm (one sigma at 25 m), a detection range of 80 m to 100 m, a 360°

[1] https://www.cyclomedia.com/

[2] https://velodynelidar.com/products/hdl-32e/

horizontal FOV, a +10° to -30° vertical FOV, and multiple returns. The sensor rotates at a rate of 20 Hz during acquisition, i.e., 0.05 seconds to complete a 360° scan of the surrounding environment. By integrating MLS sensors with IMU and GNSS into Cyclomedia's mobile data acquisition platform (Figure 34), it is also possible to obtain temporal information and match the captured MLS point cloud with the corresponding sensor trajectory. The original MLS point clouds and sensor trajectories are all provided in LAZ format[1].

The MLS data provided by Cyclomedia includes additional information such as intensity, GNSS time, and the number of returns, in addition to 3D spatial coordinate information. The sensor trajectory data records information about the sensor's position in 3D space during the scanning process. The capture points in the MLS data are in one-to-one correspondence with the sensor positions in the trajectory data by the order of storing the points, i.e., the first capture point in the MLS data corresponds to the first sensor position in the trajectory data, the second capture point corresponds to the second sensor position, and so on. Each MLS capture point and its corresponding sensor trajectory point form a line segment representing the LiDAR ray.

Figure 36: Positions of the four case sites.

[1] https://laszip.org/

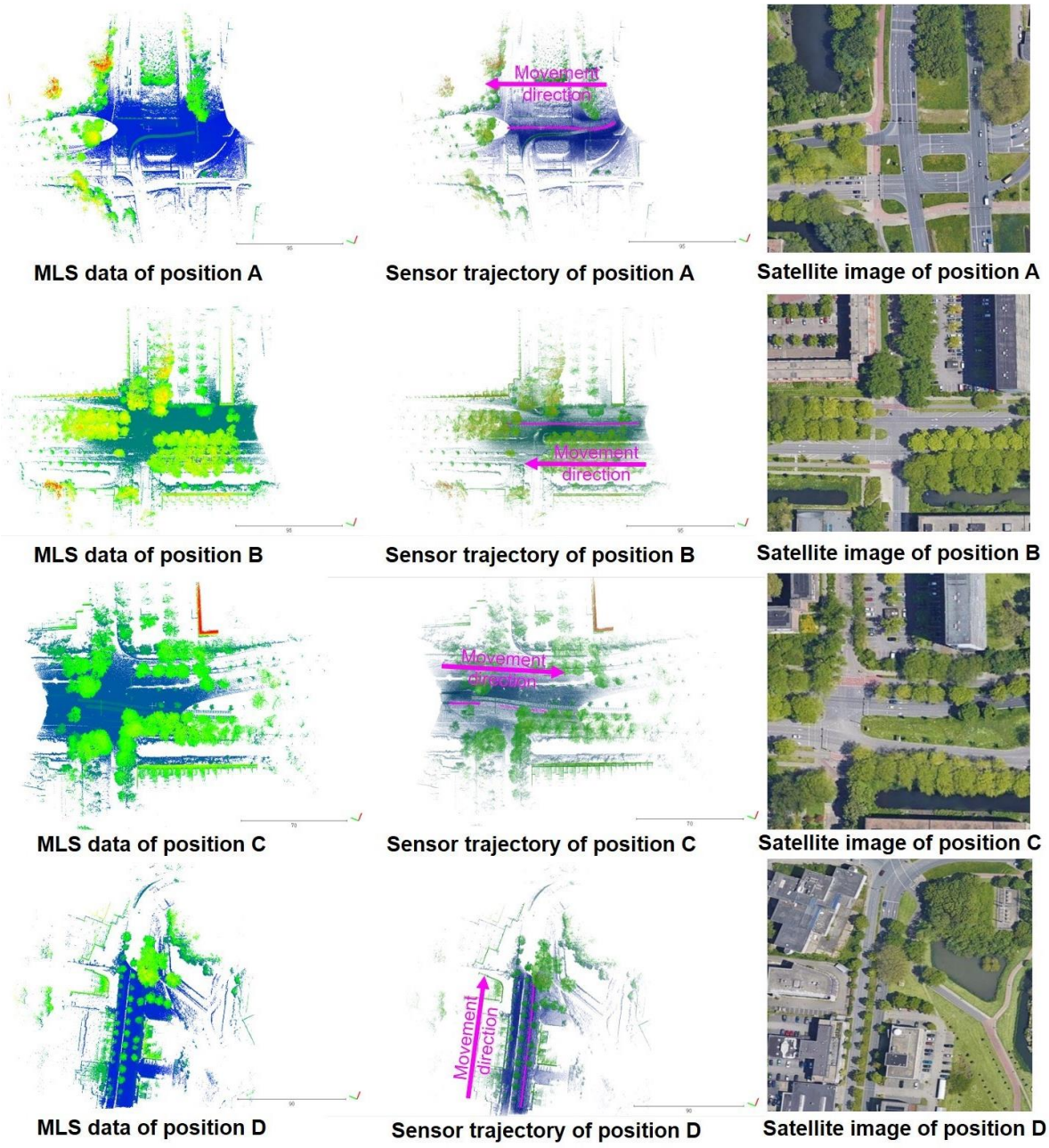


Figure 37: MLS data (left column) rendered in height, corresponding sensor trajectories (middle column) colorized in the purple, and satellite images (right column) from Google Map of the four case sites.

In the implementation phase, four case sites are selected for this research (see Figure 36 for their positions). They are all located at road junctions and have bicycle lanes, so it is easier to find different types of dynamic objects, such as vehicles and bicycles. The road network at the road junctions is more complex and influenced by traffic signals, so dynamic objects with different speeds and different moving directions can be observed in these areas. In summary, the above four positions are well suited as the case sites for this research. Each case site was scanned continuously for 10 seconds in one direction. Due to differences in the

environment, the number of points obtained for the same scan time is different for each case site and ranges from 3637969 to 4684840 (see Table 1). Figure 37 shows the MLS scan data for the four case sites and their corresponding sensor trajectory data.

Case Site	Full Name of the Case Site	Scan Time (sec)	Number of Points
Position A	The junction of Voorhofdreef, Tanthofdreef, and Kruithuisweg	10	3637969
Position B	The junction of Voorhofdreef, Menno Ter Braaklaan, and Bosboom-Toussaintplein	10	4616356
Position C	The junction of Voorhofdreef, J.J. Slauerhofflaan, and Frederik van Eedenlaan	10	4666430
Position D	The junction of Tanthofdreef and Forensenweg	10	4684840

Table 1: Point numbers of the four case sites.

5.2 Parameters

This section lists all the values of the parameters involved in the method proposed in this thesis (see Table 2) and then analyzes the basis for the values of each parameter.

Full Name of the Parameter	Parameter	Value
Time interval for data frame segmentation	t_{itvl}	0.75 sec
Octomap voxel size	$size_{voxel}$	0.2 m
Occupancy probability threshold	$thres_{occupied}$	0.7
Sensor mounting height	h_{sm}	2 m
Height restriction of large vehicles	h_{vr}	4 m
Free-point rate threshold	$thres_{fp}$	0.9
Neighborhood radius used to calculate the free-point rate	r_{ns}	1 m
Multi-return rate threshold	$thres_{mr}$	0.3
Neighborhood radius used to calculate the multi-return rate	r_{vg}	1 m
Number of nearest neighbors used to extract all vegetation points	k_{vg}	5
Number of nearest neighbors used to extract all dynamic points	$k_{do}.$	5
Minimum point number limit for dynamic objects	num_{min}	15
Threshold of seed point proportion	$thres_{sp}$	0.03

Table 2: Values of implementation parameters.

Firstly, for Octomap, a larger t_{itvl} value means that more LiDAR rays are inserted to estimate the occupancy probability more accurately, while a smaller $size_{voxel}$ allows the construction of a higher resolution voxel grid in Octomap, which in turn avoids the creation of some hybrid voxel cells (voxel cells that include both static and dynamic points). However, this also leads to larger computation and memory requirements. To strike a balance between high accuracy results and low computational and memory burden, t_{itvl} is set to 0.75 sec (14 data frames) and $size_{voxel}$ is set to 0.2 m.

For the upper and lower boundaries of the target space, h_{sm} (2 m) is obtained by directly measuring the mounting height of the sensor and h_{vr} (4 m) is taken based on the height limit of large vehicles in the Netherlands, which is got from the official document of the European Union[1].

Based on some previous studies, 0.7 is considered an appropriate threshold for occupancy probability $thres_{occupied}$ when the voxel size of Octomap takes a range of values from 0.05 m to 0.45 m (Oršulić et al., 2021; C. Wang et al., 2017). The free-point rate threshold $thres_{fp}$ is assigned to 0.9 based on the analysis results shown in Figure 29 to Figure 31. For the multiple-return rate $thres_{mr}$, some previous similar research (Ussyshkin & Theriault, 2011; Xu et al., 2012) usually set the threshold between 0.2 and 0.4 for detecting vegetation, so this parameter is set to 0.3 in this thesis.

Some influencing factors also need to be considered in setting parameters in neighboring queries. For fixed radius search, too small a radius may result in the neighborhood being easily affected by outliers, while too large a radius may easily lead to no strong spatial consistency of the points in the neighborhood. To take these two factors into account at the same time, both r_{ns} and r_{vg} are set to 1 m. For the KNN search in the vegetation removal and dynamic object extraction steps, too small a value of k may fail to extract the complete target object, and too large a value of k may identify several neighbor objects as one object and then extract them together. To balance these two points, both k_{vg} and k_{do} are set to 5.

In the final dynamic object extraction, two constraints are used to filter the valid dynamic objects: (1) The minimum number of points within the object num_{min} must be greater than k_{do} to remove some abnormal objects that are too small. It is set to 15 in the implementation of this research. (2) The proportion of seed points within the object $thres_{sp}$ must be greater than 0.03 to prevent the false detection of large-area vegetation caused by a few noise points.

5.3 Tools

This section begins with a description of the tools used to implement the proposed method and then concludes with a summary of the collaboration pipeline between these tools.

[1] https://eur-lex.europa.eu/eli/dir/1996/53/2019-08-14

The main workflow of the method proposed in this thesis is based on a C++ implementation. The C++ code was run on an AMD Ryzen 9 CPU 3.30 GHz with 16 GB RAM. The implementation of Octomap in this study relies on the Octomap Library developed by Kai M. Wurm and Armin Hornung from the University of Freiburg[1]. Other point cloud operations such as building KD-trees, spatial neighborhood queries, etc. are dependent on PCL[2], which is a large-scale open-source point cloud processing project.

The format conversion before data processing and the visualization of the final output is done in CloudCompare[3], which is an open-source point cloud viewing, processing, and editing software. In terms of data transfer convenience, the original MLS point cloud data and sensor trajectory data are stored in LAZ format and need to be converted to PCD format in CloudCompare for reading by C++ libraries such as PCL. The final implementation results are also visualized with built-in display and rendering modules of CloudCompare.

Figure 38 illustrates the collaboration pipeline between the above-mentioned tools.

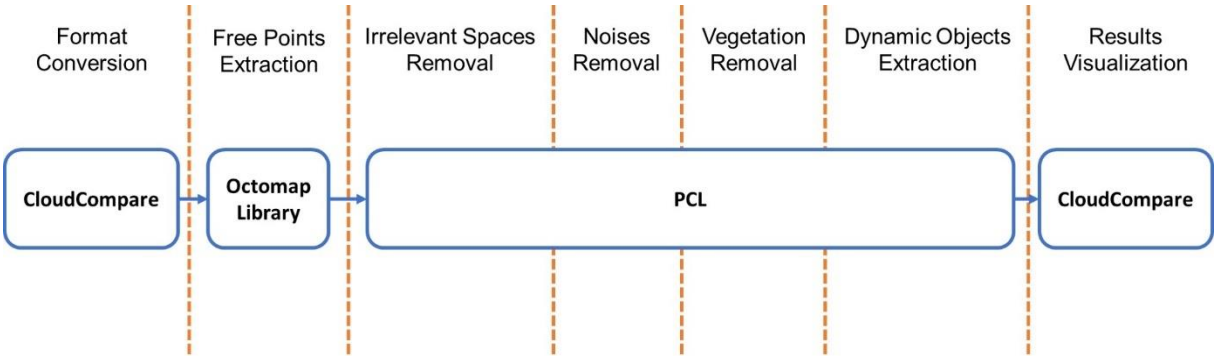


Figure 38: The collaboration pipeline between all used tools in this research.

[1] https://github.com/OctoMap/octomap
[2] https://github.com/PointCloudLibrary/pcl
[3] https://github.com/CloudCompare/CloudCompare

6. Results and Discussion

This chapter first shows the implementation results of four case sites (Section 6.1), then assesses the detection accuracy of the four implementation results, after that discusses the factors that affect the accuracy of the implementation results (Section 6.2). Finally, this chapter analyzes the proposed method in terms of running time and memory consumption and compares its performance with the original Octomap method.

6.1 Implementation Results

Figure 39 to Figure 42 show the dynamic object detection and extraction results for positions A to D. All points in the high-altitude space and ground area are labeled as static points. So only the detection and extraction results of the points in ROI are shown in these four figures.

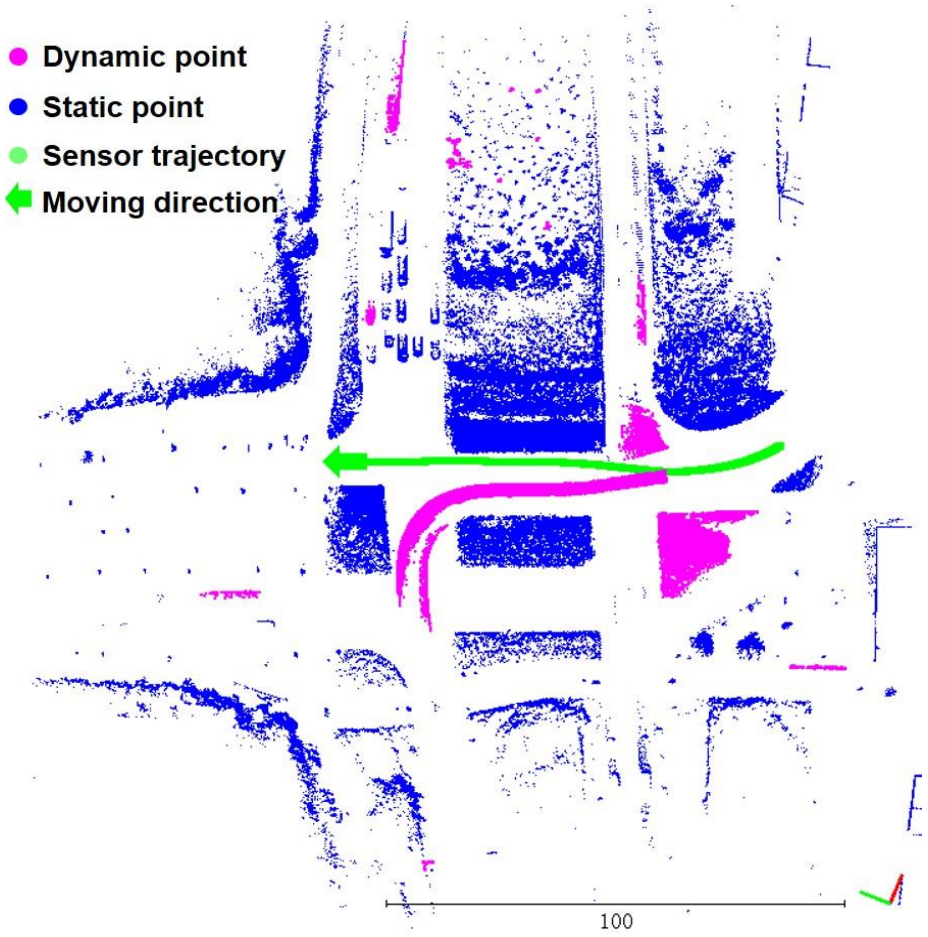


Figure 39: The detection and extraction results of dynamic objects with corresponding sensor trajectory in position A.

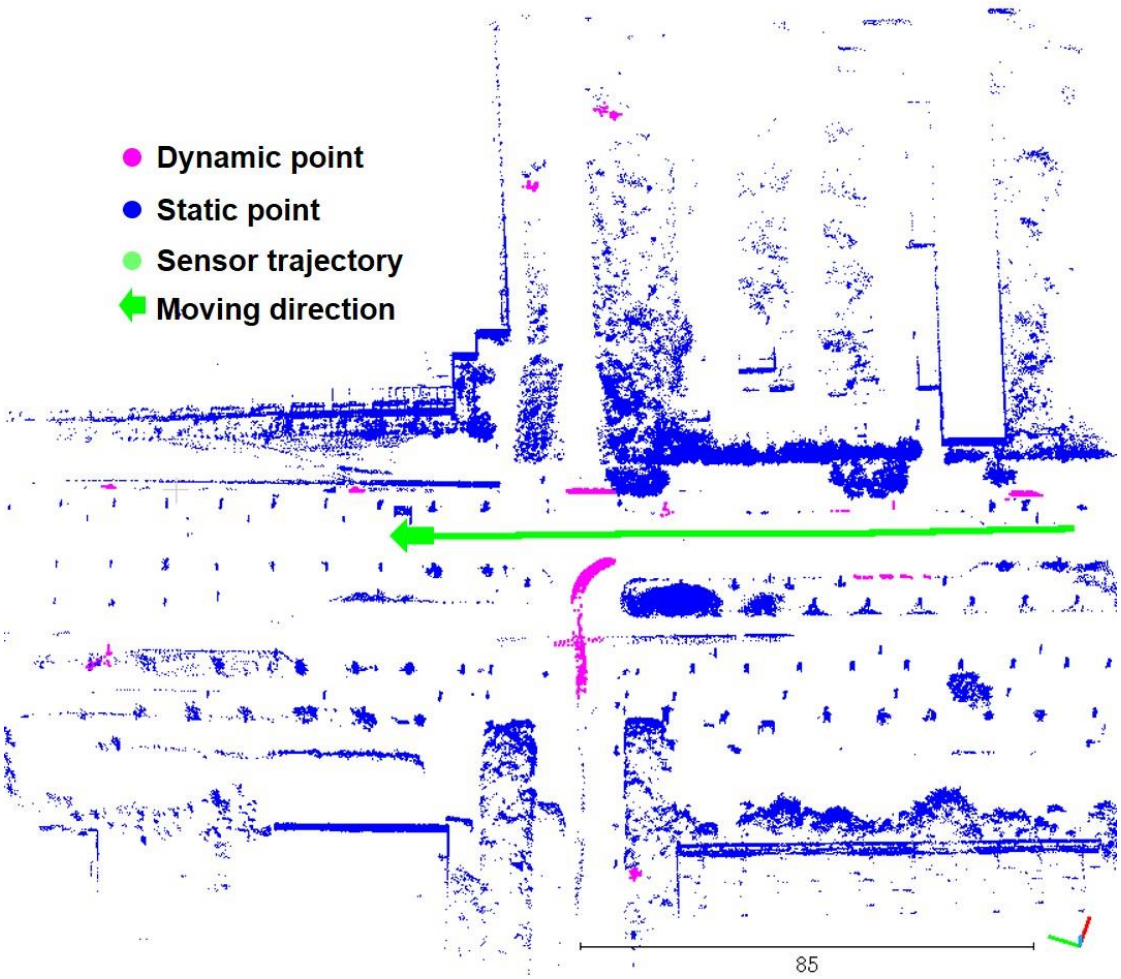


Figure 40: The detection and extraction results of dynamic objects with corresponding sensor trajectory in position B.

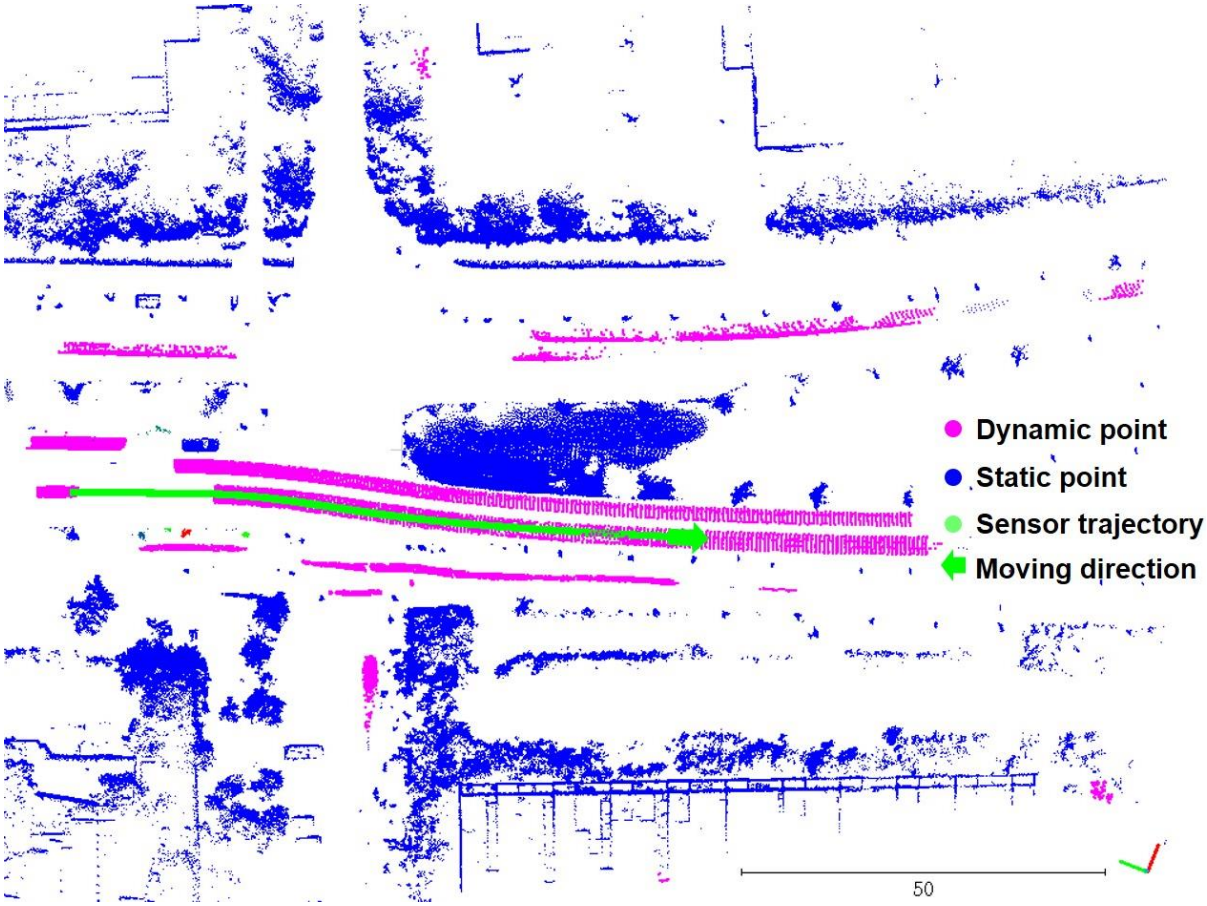


Figure 41: The detection and extraction results of dynamic objects with corresponding sensor trajectory in position C.

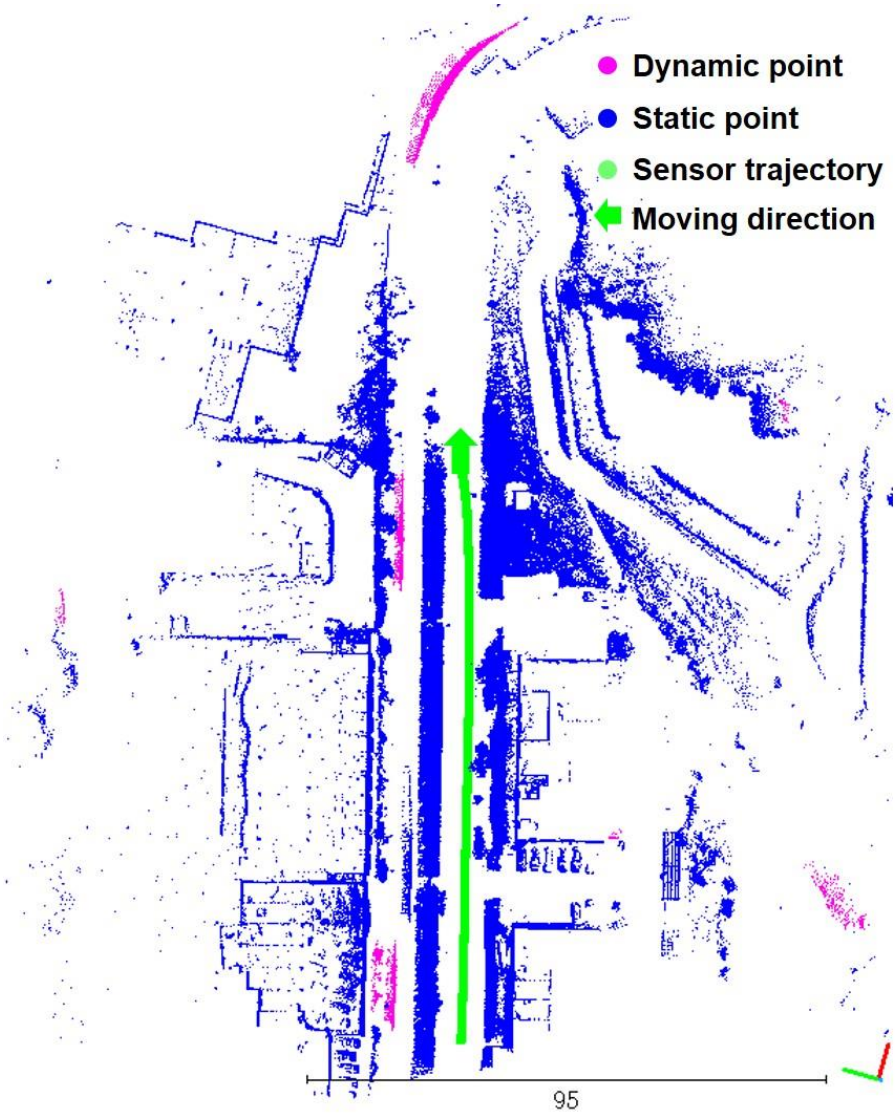


Figure 42: The detection and extraction results of dynamic objects with corresponding sensor trajectory in position D.

6.2 Performance Analysis

This section first evaluates the accuracy of dynamic object detection and extraction using confusion matrices (Subsection 6.2.1). Then, the factors that affect the accuracy of the results are analyzed (Subsection 6.2.2). Finally, this section compares the advantages of parallel computing over serial computing in terms of runtime (Subsection 6.2.1).

6.2.1 Accuracy Assessment

Dynamic object detection and extraction is essentially a point cloud binary classification issue. Therefore, this subsection first manually labels the dynamic objects on the data of the four case sites, and then builds confusion matrices for each of the four implementation results. Finally, four indicators are introduced to evaluate the accuracy of the proposed method: user's accuracy, producer's accuracy, overall accuracy, and Cohen's kappa coefficient. The user's accuracy here is the percentage of points in each category (dynamic object and static object) of the implementation result that are correctly detected as dynamic points or static points. The producer's accuracy infers the percentage of points in each category of the ground truth (manually labelled data) that are correctly detected as dynamic points or static points. The overall accuracy means the sum of correctly detected dynamic or static points as a percentage of all MLS capture points. The kappa coefficient is an assessment of the consistency of the detection results, and its value is generally between 0 and 1, with closer to 1 indicating higher detection accuracy. Compared to the overall accuracy, the kappa coefficient considers the imbalance between objects. Table 3 shows the standard 2-by-2 confusion matrix with the corresponding user's accuracy and producer's accuracy. Table 4 shows the equations of the overall accuracy and kappa coefficient.

	Dynamic Points in Ground Truth	**Static Points in Ground Truth**	**User's Accuracy (UA)**
Dynamic Points in Implementation Result	X_{11}	X_{12}	$\frac{X_{11}}{X_{11}+X_{12}} \cdot 100\%$
Static Points in Implementation Result	X_{21}	X_{22}	$\frac{X_{22}}{X_{11}+X_{12}} \cdot 100\%$
Producer's Accuracy (PA)	$\frac{X_{11}}{X_{11}+X_{21}} \cdot 100\%$	$\frac{X_{22}}{X_{12}+X_{22}} \cdot 100\%$	

Table 3: The standard 2-by-2 confusion matrix with the corresponding user's accuracy and producer's accuracy.

Overall Accuracy (OA)	Kappa Coefficient (KC)
$\frac{X_{11}+X_{22}}{X_{11}+X_{21}+X_{12}+X_{22}} \cdot 100\%$	$\frac{OA-\frac{(X_{11}+X_{21})\cdot(X_{11}+X_{12})+(X_{21}+X_{22})\cdot(X_{12}+X_{22})}{(X_{11}+X_{21}+X_{12}+X_{22})^2}}{1-\frac{(X_{11}+X_{21})\cdot(X_{11}+X_{12})+(X_{21}+X_{22})\cdot(X_{12}+X_{22})}{(X_{11}+X_{21}+X_{12}+X_{22})^2}}$

Table 4: The equations of the overall accuracy and kappa coefficient.

Table 5 and Table 6 show the confusion matrices and values of the corresponding four accuracy assessment indicators for positions A to position D obtained from the template provided in Table 3 and Table 4. The four case sites have very high user's accuracy and producer's accuracy for static object detection (>99.9%). Since dynamic and static objects in MLS data are usually very imbalanced (i.e., the number of static points is much larger than the number of dynamic points), the overall accuracy is more likely to be affected by the detection accuracy of static objects. Therefore, the overall accuracy of all four case sites is also very high (>99.5%). However, for the producer's accuracy and user's accuracy as well as the kappa coefficients of the dynamic object detection focused by this thesis, a large difference is shown in the implementation results of these four case sites. The implementation result of position C has the highest dynamic object detection user's accuracy (94.429%), producer's accuracy (93.753%), and kappa coefficient (0.936). While position B has the lowest user's accuracy (75.361%) for dynamic object detection, which means that many static objects are incorrectly detected and extracted as dynamic objects. The lowest producer's accuracy (74.345%) and kappa coefficient (0.674) of dynamic object detection are found at position D, which means that many dynamic objects are not successfully detected and extracted. The low kappa coefficient of position D is mainly caused by the low percentage of dynamic points in the ground truth. The percentage of dynamic points in position D, which has the lowest kappa coefficient, is only 0.196% and in position B, which has the second-lowest kappa coefficient, is 0.274%. In contrast, the two case sites with high kappa coefficients, position A and position C, have relatively high dynamic point percentages (1.591% and 0.766%). Therefore, a few incorrectly detected points more significantly decrease the kappa coefficient of position D compared to the other three case sites.

		Dynamic Points in Ground Truth	Static Points in Ground Truth	User's Accuracy (UA)
Position A	**Dynamic Points in Implementation Result**	57705	15408	78.926%
	Static Points in Implementation Result	188	3564668	99.995%
	Producer's Accuracy (PA)	99.675%	99.570%	

Position B	**Dynamic Points in Implementation Result**	11002	3597	75.361%
	Static Points in Implementation Result	1665	4600092	99.964%
	Producer's Accuracy (PA)	86.856%	99.922%	
Position C	**Dynamic Points in Implementation Result**	33529	1978	94.429%
	Static Points in Implementation Result	2234	4628689	99.952%
	Producer's Accuracy (PA)	93.753%	99.957%	
Position D	**Dynamic Points in Implementation Result**	6813	1951	77.738%
	Static Points in Implementation Result	2351	4673725	99.950%
	Producer's Accuracy (PA)	74.345%	99.958%	

Table 5: The confusion matrix with the corresponding user's accuracies and producer's accuracies of the four case sites.

The weighted average producer's and user's accuracies for dynamic object detection and extraction among these four case sites are 88.004% and 82.624%, respectively. The weighted average overall accuracy is 99.833%. For the producer's accuracy of dynamic object detection, its weight is the number of dynamic points in the ground truth in each case site. For the user's accuracy of dynamic object detection, its weight is the number of dynamic points in the implementation result in each case site.

Position	Overall Accuracy (OA)	Kappa Coefficient (KC)
A	99.571%	0.878
B	99.886%	0.780
C	99.910%	0.936
D	99.908%	0.674
Weighted Average Value	99.833%	0.814

Table 6: The overall accuracies and kappa coefficients of the four case sites.

6.2.2 Influence Factors

This subsection analyzes which factors significantly affect the detection results and to which factors the proposed method is insensitive, based on the visualization of the implementation results and the accuracy assessment.

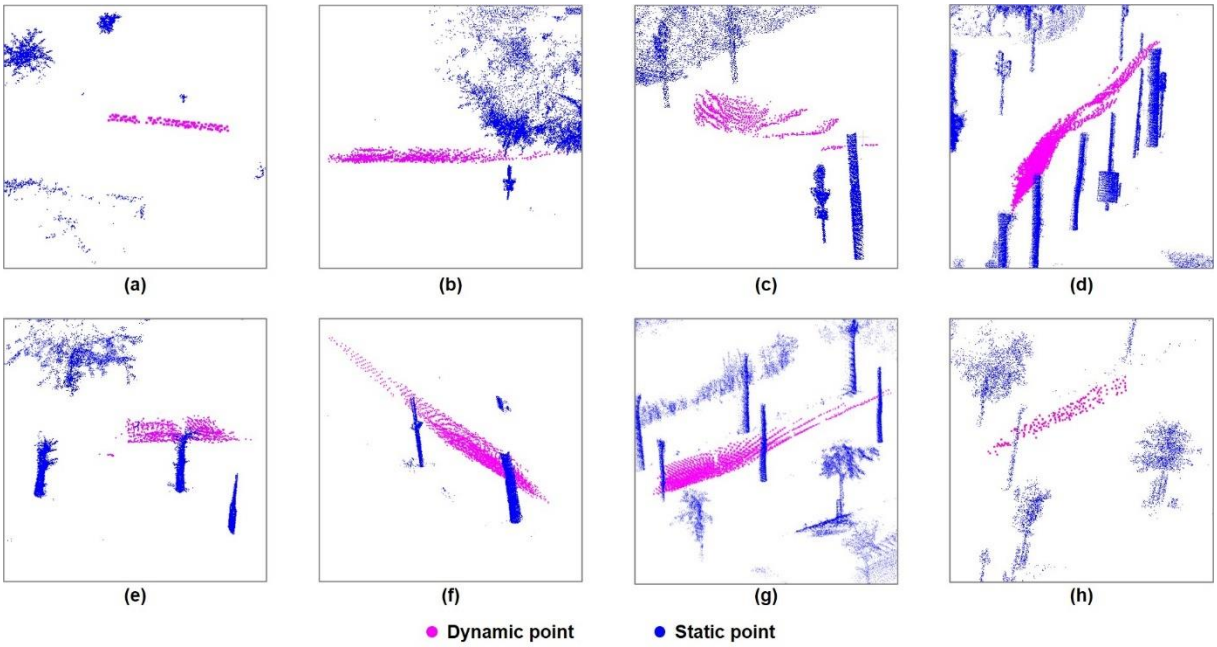


Figure 43: The detected moving bicycles.

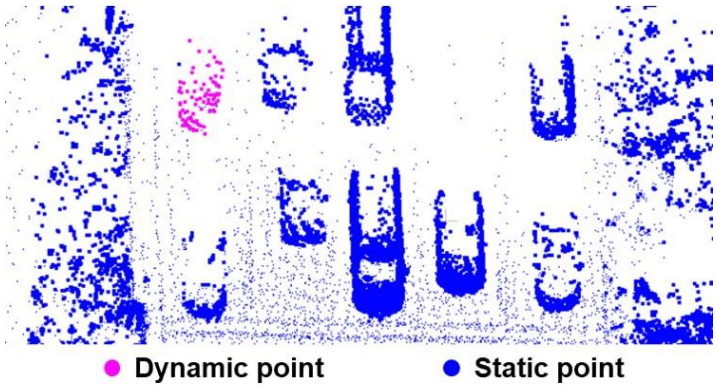


Figure 44: The detected braking vehicle.

In previous research on dynamic object detection, the speed of the object is an important factor that affects the detection results. The detection of low-speed objects can be more challenging than high-speed objects (Dewan et al., 2016). However, the speed of dynamic objects does not significantly affect the results of the proposed method in the observed results. Most low-speed objects such as moving bicycles (Figure 43) and vehicles that are braking can be correctly detected (see the upper left corner of Figure 44). The size of the dynamic objects is another factor of concern. In the four case sites of this study, most of the small dynamic objects (e.g., bicycles in Figure 43) are identified. However, no pedestrians are found in these four case sites, so whether this method is also applicable to detecting pedestrians of small size and low speed will need to be further explored in the future.

Whether the movement direction of the dynamic object with respect to the MLS sensor affects the detection results is also analyzed. Since all four case sites are located near road junctions, objects moving along different directions can be observed. Figure 45 illustrates several objects that move in directions different from the MLS sensor. The dynamic objects in Figure 45.(a) and Figure 45.(b) are moving perpendicular to the MLS sensor trajectory while the dynamic objects in Figure 45.(c) and Figure 45.(d) are moving in the opposite direction of the sensor motion. These dynamic objects do not fail to be detected due to their different moving directions. Another noticeable phenomenon is that the dynamic object does not produce a significant lateral stretch when it moves perpendicular to the sensor trajectory. Its shape is also not compressed when it moves in the opposite direction to the sensor. For the four case sites, the ghost trail effect (i.e., being stretched along its own direction of motion) is observed regardless of the direction of motion of the dynamic object.

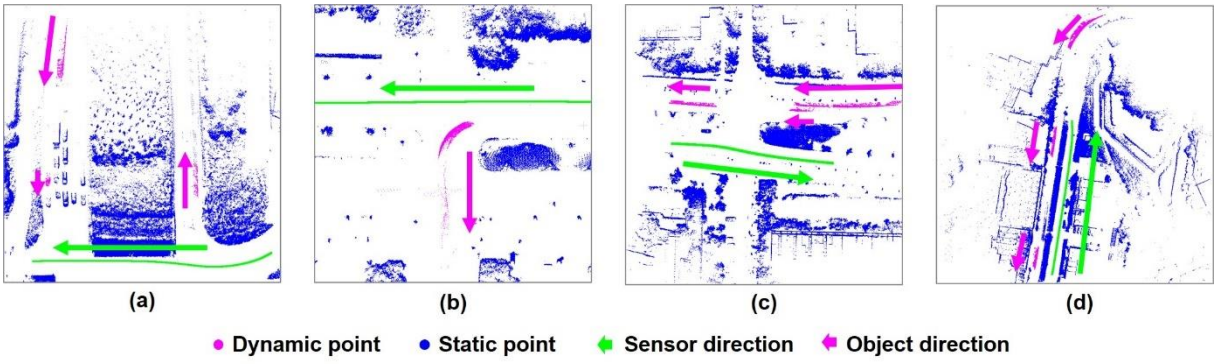


Figure 45: Objects moving in directions different from the MLS sensor.

But some factors can significantly affect the performance of the proposed method. Two of the most important factors are the performance of the vegetation and noise removal methods since most static objects that are incorrectly detected as dynamic objects are vegetation (see Figure 46). These mis-detected vegetation are mainly caused by the residual vegetation points and noisy points in the free points. The performance of the ground filtering method also affects the detection and extraction results. The ground extraction operation in Section 4.2 assumes that the ground is a flat surface and does not vary significantly in height within a certain range, but some surfaces in the real world do not conform to this assumption. There are two ground areas in position A that are not successfully removed due to their uneven surfaces. These two ground areas are detected as dynamic objects (see Figure 47.(a)). Only part of the points in these two ground areas are detected as dynamic points and do not leave significant voids in the ground. Therefore, these mis-detected ground points do not break the ground connectivity (Figure 47.(b)). However, they inevitably change some geometric

properties of these ground areas, such as local ground height and point density. In addition, the mis-detected objects include a small number of remnant buildings and pole-like objects such as streetlights, traffic lights, and traffic signs (Figure 48).

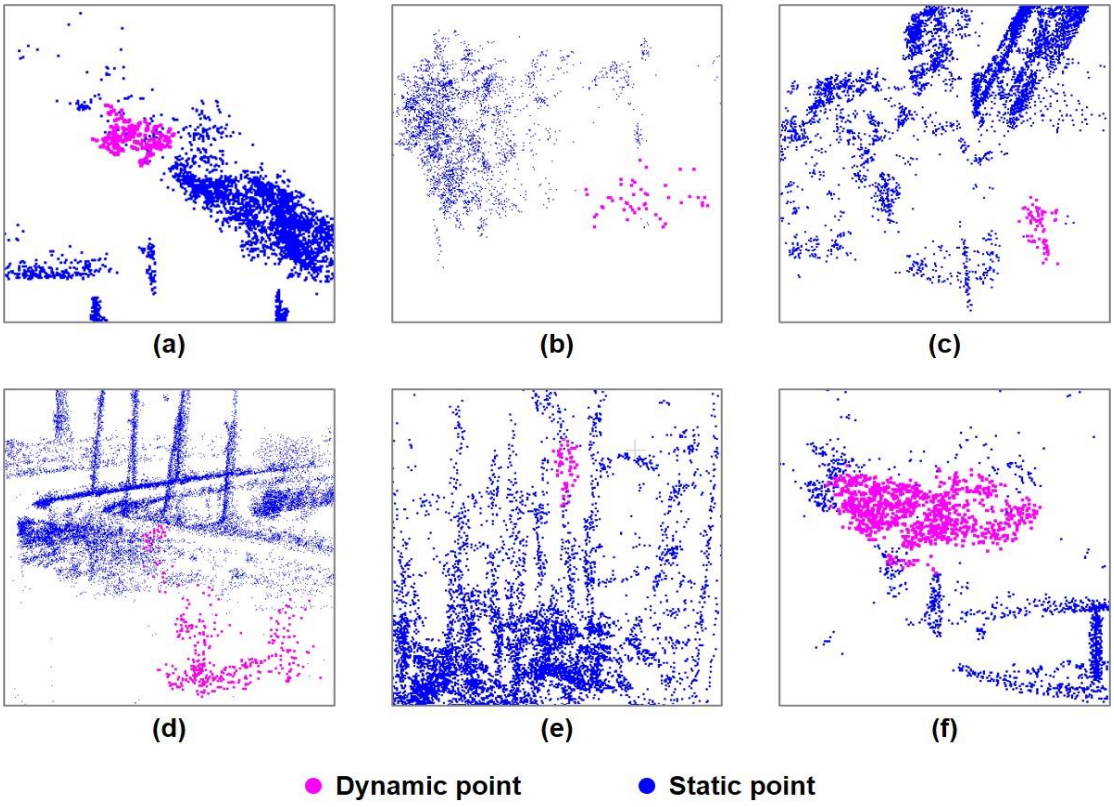


Figure 46: The mis-detected vegetation.

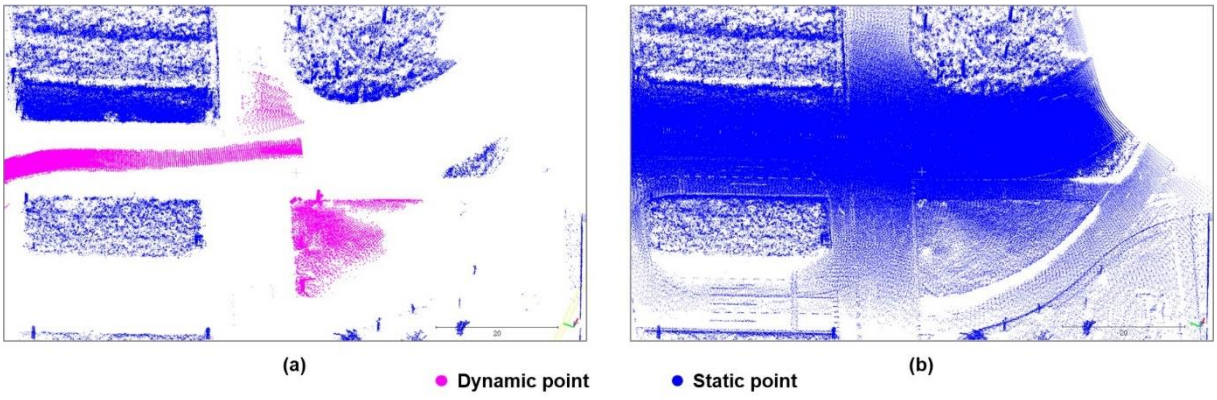


Figure 47: The mis-detected ground points (a) and their corresponding ground surface (b).

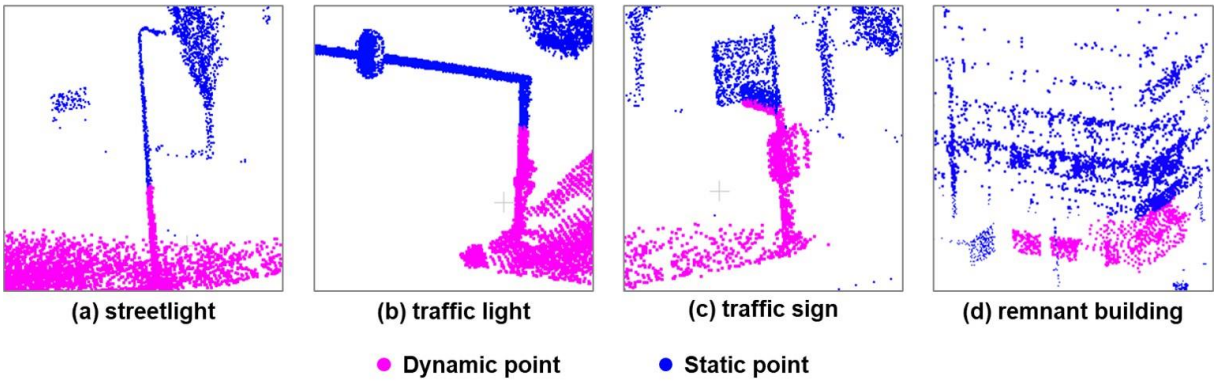


Figure 48: The mis-detected remnant building and various mis-detected pole-like objects.

The main reason why some dynamic objects in the implementation results are not successfully detected is the distance from the dynamic object to the MLS sensor. When the dynamic object is too far away from the MLS sensor, it generates many problems such as low point density (see Figure 49), which can lead to incomplete detection or failed detection of dynamic objects. Conversely, the detection accuracy is higher if the dynamic objects are closer to the MLS sensor. The main reason for the high detection accuracy of the implementation result in position C is that most of its dynamic objects are very close to the MLS sensor and have similar motion trajectories to the MLS sensor. But this can be solved by adding previous or next data frames if this object is located on the sensor's trajectory.

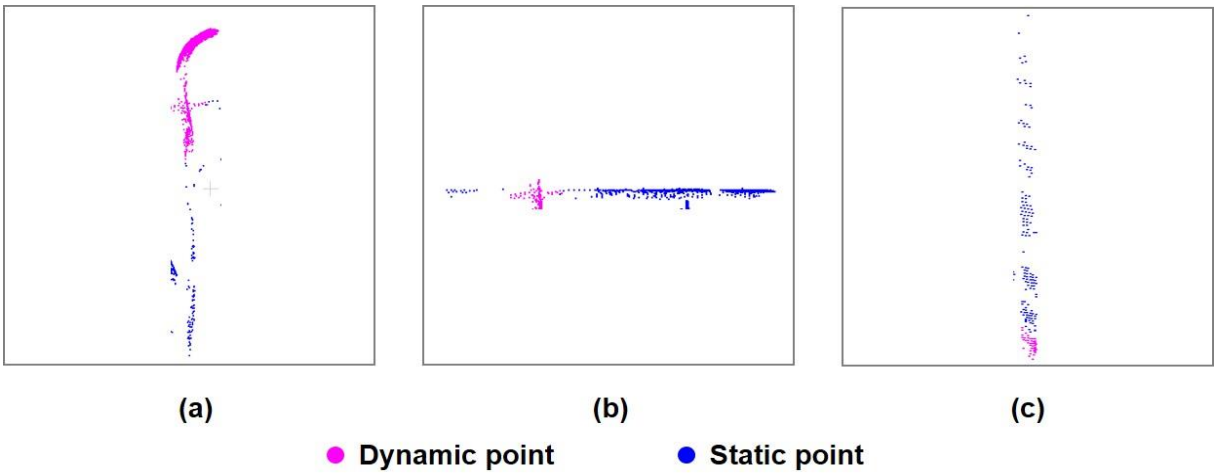


Figure 49: Sparse dynamic objects that are not detected completely.

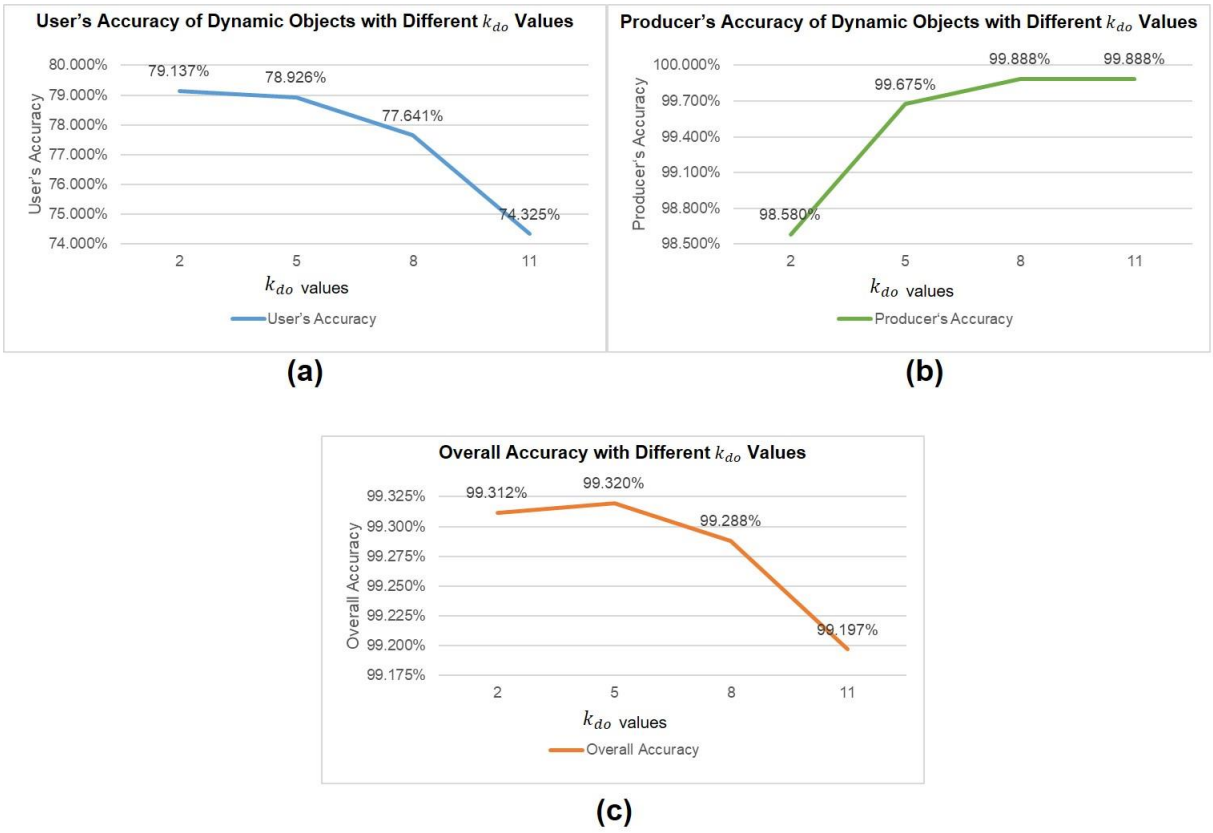


Figure 50: The user's accuracy (a) and the producer's accuracy (b) of dynamic objects, and the overall accuracy (c) of the whole results using different k_{do} values.

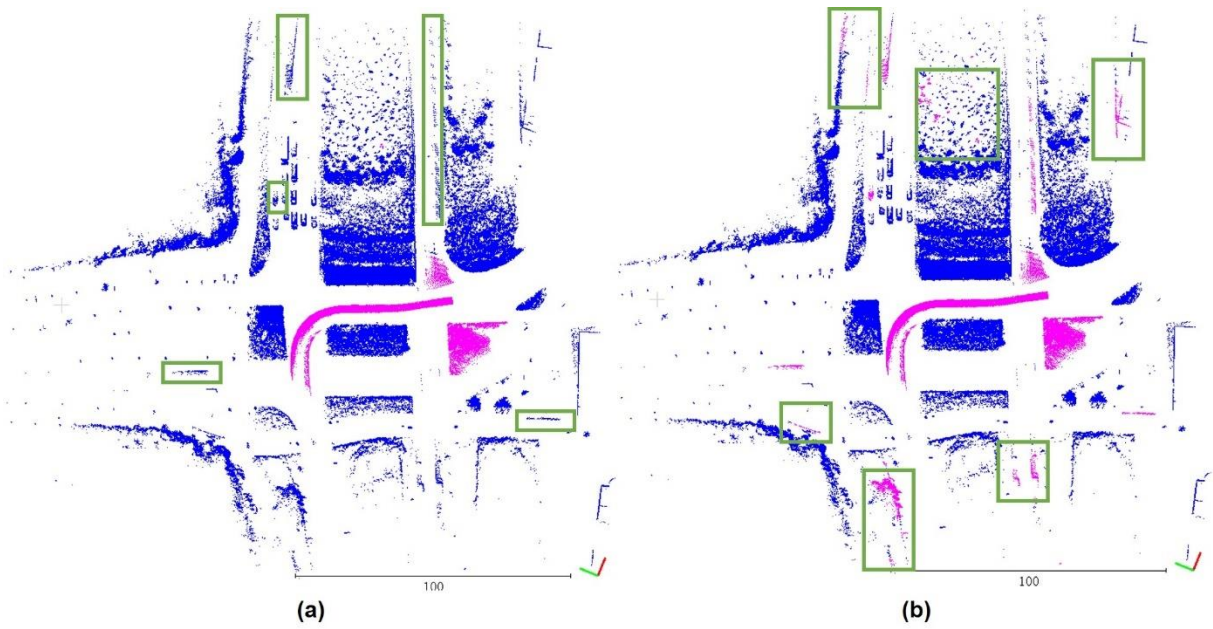


Figure 51: The detection results of position A when k_{do} is set to 2 (a) and 11 (b).

The last influencing factor is the choice of parameter values. In this thesis, the number of nearest neighbors used to extract dynamic objects (k_{do}) is chosen as a test case among all parameters using captured data from position A. This value is the core parameter for the last part of the whole method, so the impact of adjusting this parameter value can be reflected very intuitively in the final detection results. This value is the core parameter for the last step of the whole method, so the impact of adjusting this parameter value can be more directly and intuitively reflected in the final detection results. In the test, the values of k_{do} are set to 2, 5, 8, and 11, respectively. The results showed that as the value of k_{do} increased, the user's accuracy decreased from 79.137% to 74.325% (Fugure 50.(a)), while the producer's accuracy increased from 98.580% to 99.888% (Fugure 50.(a)). Setting the k_{do} value too small will result in many dynamic points not being recognized correctly (see the green boxes in Figure 51.(a)), while setting it too large will incorrectly recognize many static points as dynamic points (see the green boxes in Figure 51.(b)). So finally the value of k_{do} is set to 5 in this thesis to obtain the highest overall accuracy (99.320%, see Figure 50.(c)).

6.2.3 Running Time and Memory Consumption

This subsection first analyzes the running time required for the implementation of the proposed optimized Octomap method and the acceleration effect using parallel computing. Then the time and memory spent in extracting free points between this method and the original Octomap method are compared.

One of the challenges in point cloud processing is the conflict between massive amounts of data and efficient data processing (Hu et al., 2013). Therefore, for point cloud data, a parallel computation implements a more efficient point cloud processing than serial computation using a single thread (Najdataei et al., 2018; Sugumaran et al., 2011).

The parallel computation in this implementation is done by asynchronous tasks created by *std::future* and *std::async* in C++ with four threads enabled. The proposed optimized Octomap method splits the original MLS data into several data frames based on the timestamp. The data frames are independent of each other, so that most operations of the proposed optimized Octomap method in this thesis are compatible with parallel computation to improve efficiency. However, due to the limitations of C++'s built-in *std::future* and *std::async* functions, the two operations of merging free points extracted from different Octomaps and removing redundant points are not parallelized in this method.

Case Site	Point Number	Optimized Octomap 1 Thread (sec)	Optimized Octomap 2 Threads (sec)	Optimized Octomap 3 Threads (sec)	Optimized Octomap 4 Threads (sec)	Total Speed-up
Position A	3637969	1042	765.5	637	525	98.476%
Position B	4616356	1415	1132	977.5	882	60.488%
Position C	4666430	1331	1033	904	649	105.085%
Position D	4684840	1342	1143	1051	870	54.253%

Table 7: Running time of the proposed optimized Octomap based on different number of threads in four case sites.

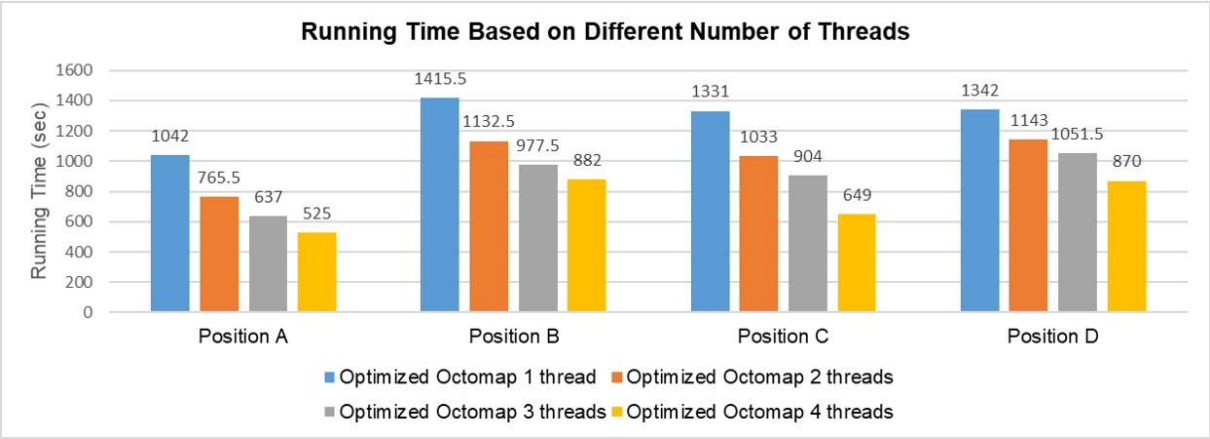


Figure 52: Running time of the optimized Octomap based on different number of threads in four case sites.

Figure 52 and Table 7 show the running time based on different numbers of threads for the four case sites. All four case sites were scanned for 10 seconds. Their number of MLS points collected range from 3637969 to 4684840 due to differences in the spatial environment. Four threads are on average 79.545% more efficient than a single thread. The results show that enabling multiple threads for parallel computation can improve the efficiency of the proposed optimized Octomap method compared to single-threaded serial computation in the same running environment.

Then the proposed optimized Octomap method is compared with the original Octomap method to check whether it has better performance in terms of computational efficiency as

well as memory consumption. Since the operations in Section 4.2 to Section 4.5 are not related to Octomap, only comparing the running time and the maximum memory consumption of the two methods when constructing probabilistic voxel grids and extracting the free points (i.e., the operations described in Section 4.1).

Case Site	Point Number	Original Octomap (sec)	Optimized Octomap 1 Thread (sec)	Optimized Octomap 2 Threads (sec)	Optimized Octomap 3 Threads (sec)	Optimized Octomap 4 Threads (sec)
Position A	3637969	973.5	721.5	507.5	399	302.5
Position B	4616356	1562.5	958.5	725.5	535.5	385.5
Position C	4666430	1361.5	995.5	755.5	627.5	403.5
Position D	4684840	791	738	612.5	503	410.5

Table 8: Running time of voxel grids generation and free points extraction.

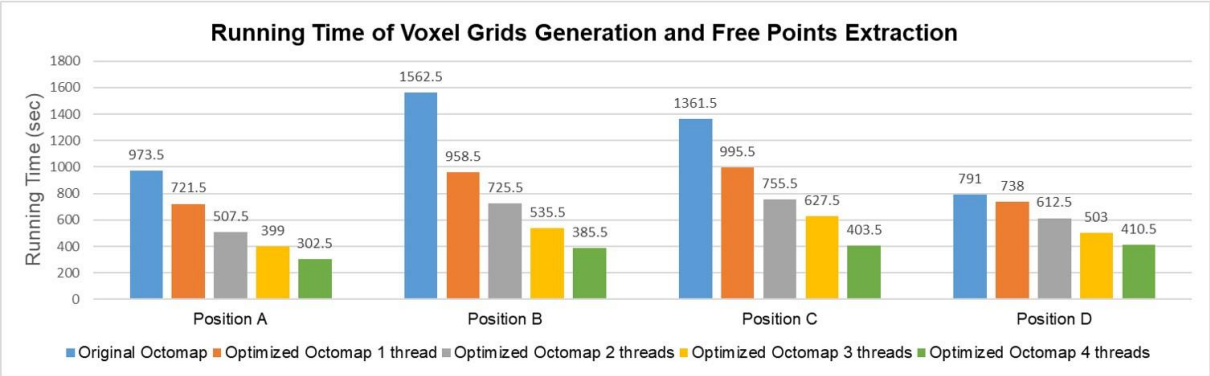


Figure 53: Running time of voxel grids generation and free points extraction.

As shown in Figure 53 and Table 8, the proposed optimized Octomap method (under a single thread) is computationally more efficient than the original Octomap method in generating probabilistic voxel grids and in extracting free points. As more threads are enabled, the computational efficiency advantage of the proposed optimized Octomap method becomes more apparent. Table 9 shows that the proposed method accelerates on average 35.472% over the original Octomap in the four case sites when single thread is enabled. With 2 to 4

threads enabled, the proposed optimized Octomap method accelerates 79.137%, 127.499%, and 214.313%, respectively, over the original Octomap method.

Case Site	1 thread	2 threads	3 threads	4 threads
Position A	34.927%	91.823%	143.985%	221.818%
Position B	63.015%	115.369%	191.783%	305.318%
Position C	36.765%	80.212%	116.972%	237.423%
Position D	7.182%	29.143%	57.256%	92.692%
Average Speed-up	35.472%	79.137%	127.499%	214.313%

Table 9: The computational speed-up of the proposed optimized Octomap method compared to the original Octomap method.

From Figure 54, Table 10, and Table 11, the maximum memory consumption of the proposed optimized Octomap method is only on average 42.437% of that of the original Octomap with a single thread, and the maximum memory consumption is close to that of the original Octomap method only when 3 threads are enabled. Therefore this designed method is more memory friendly than the original Octomap method.

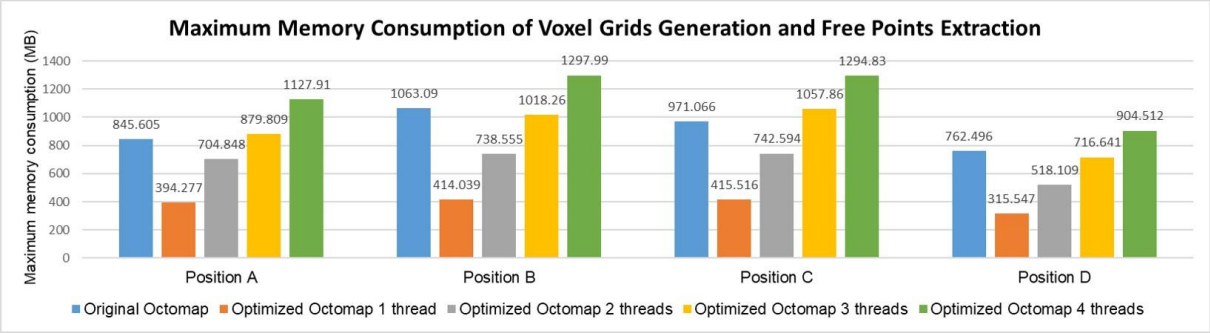


Figure 54: Maximum memory consumption of voxel grids generation and free points extraction.

Case Site	Original Octomap	Optimized Octomap 1 thread (MB)	Optimized Octomap 2 threads (MB)	Optimized Octomap 3 threads (MB)	Optimized Octomap 4 threads (MB)
Position A	845.605	394.277	704.848	879.809	1127.91
Position B	1063.09	414.039	738.555	1018.26	1297.99
Position C	971.066	415.516	742.594	1057.86	1294.83
Position D	762.496	315.547	518.109	716.641	904.512

Table 10: Maximum memory consumption of voxel grids generation and free points extraction.

Case Site	Optimized Octomap 1 thread	Optimized Octomap 2 threads	Optimized Octomap 3 threads	Optimized Octomap 4 threads
Position A	46.627%	83.354%	104.045%	133.385%
Position B	38.947%	69.472%	95.783%	122.096%
Position C	42.790%	76.472%	108.938%	133.341%
Position D	41.383%	67.949%	93.986%	118.625%
Average Values	42.437%	74.312%	100.688%	126.862%

Table 11: The maximum memory consumption of the proposed optimized Octomap method as a percentage of the original Octomap.

7. Conclusions and Future Work

This chapter first reviews the research questions and gives the corresponding conclusions in Section 7.1, then lists contributions of this thesis in Section 7.2, and finally analyzes some future work in Section 7.3.

7.1 Research Conclusions

This thesis focuses on one main research question: **How to detect and remove dynamic objects from MLS data?** To address this main research question, this thesis proposes a method to detect and remove dynamic objects in MLS data using Octomap. The proposed method first splits the original MLS data into multiple data frames based on timestamps. Each data frame and its neighbor data frames are merged into one group and inserted into an independent Octomap, which generates multiple smaller Octomaps to avoid generating a very huge Octomap. The free points can be obtained from all Octomaps by setting an occupancy probability threshold. Then, the ROI is reduced by removing the free points located in the ground and the high-altitude space. Next, the vegetation points and noise in the free points are removed by calculating the free-point rate and the multi-return rate. Finally, the remaining free points are used as seed points to detect and extract dynamic objects from the original MLS data using KNN spatial search. All operations of this proposed method are compatible with parallel computation to improve efficiency, except for the two operations of merging free points extracted from different Octomaps and removing redundant points.

The method is tested with four case sites and its producer's and user's weighted average dynamic object detection and extraction accuracies are 88.004% and 82.624%, respectively. The weighted average overall accuracy is 99.833%. The implementation results and accuracy assessment demonstrate that the proposed method can be effectively applied to dynamic object detection and extraction tasks in MLS data sets and has advantages over the original Octomap method in terms of computational efficiency and memory consumption through parallel computing.

The research results of this thesis also answer the other sub-questions listed in Section 1.2:

(1) How to detect and remove dynamic objects and avoid residue?

The dynamic object detection in this thesis is based on Octomap's free point extraction. The LiDAR rays are first reconstructed in Octomap, and the spatial conflicts between the rays are counted to calculate the occupation probability of each voxel space. The points located in the low occupancy probability space (i.e., free space) are free points. Then for removing noise and vegetation points from free points, the free-point rate and multi-return rate are calculated by using a fixed radius spatial search. After that dynamic object detection and extraction are achieved by using KNN spatial search with filtered free points as seed points.

The weighted average producer's accuracy for dynamic object detection is 88.004% in the implementation results of the four case sites, which means that the proposed method hardly produces incomplete dynamic object extraction. The very few incomplete dynamic object detection and extraction occur mainly on sparse objects far from the MLS sensor.

(2) How to avoid detecting and removing static environment objects?

The free points in Octomap are used as seed points for detecting and extracting dynamic objects in this method. However, the initial free points are inevitably mixed with some static points. So, this method first delimits the ROI to remove the static-free points located on the ground and at high altitudes. Then the static-free points are further removed using vegetation and noise removal operations so that the final free points include only dynamic points as much as possible.

The weighted average user's accuracy for dynamic object detection is 82.624% in the implementation results of the four case sites. This means that most of the detection results are correct for dynamic objects, including only a small number of static points, such as vegetation, remaining buildings, streetlights, traffic lights, and traffic signs.

(3) What factors affect the detection results?

The main cause of misdetection is the performance of the vegetation and noise removal methods. The minor cause is the performance of the ground filtering method. Remaining ground points, vegetation points, noise points, and pole-like objects in the free points can cause some static objects to be incorrectly detected as dynamic objects.

The main reason why dynamic objects are not successfully detected is the distance of the object from the MLS sensor. A dynamic object that is too far from the MLS sensor causes its captured points to be too sparse and thus difficult to be detected completely.

In addition, the choice of parameter values also affects the final test results. For example, the number of nearest neighbors used to extract dynamic objects (k_{do}) is set to a very small value can lead to some dynamic objects not being detected correctly, while too large a k_{do} value can lead to misdetection.

Compared with the above factors, the size, speed, and movement direction of the dynamic object are not observed to have a significant effect on the detection results in the implementation.

(4) How to use MLS sensor trajectory to assist detection and removal operations?

In this method, the MLS sensor trajectory has two main roles: (1) to reconstruct the LiDAR rays in Octomap with corresponding MLS capture points, and (2) to delimit the upper and lower boundaries of the ROI by obtaining the MLS sensor height from the trajectory.

(5) What types of objects often lead to misdetection?

The most common mis-detected objects in the implementation results are vegetation, and other less common mis-detected objects include some small ground areas, remnant buildings, pole-like objects such as streetlights, traffic lights, and traffic signs.

(6) How to reduce the computational time and the memory requirement for processing large-scale data?

Computational efficiency and memory requirement have been major challenges for MLS point cloud data processing. This problem was further exacerbated in the previous Octomap-based dynamic object detection approaches. So, the proposed method takes three initiatives: (1) Reduce the computation and memory requirements by generating multiple smaller Octomaps to avoid generating a very huge Octomap. (2) Obtain a smaller ROI by removing the ground area and the high-altitude space to reduce the computation cost in subsequent steps. (3) Most steps of the whole processing workflow can be accelerated with parallel computing.

Using parallel computing, the proposed method with 4 threads enabled achieves an average 73.496% improvement in computational efficiency over single thread. Compared to the original Octomap method, the proposed method has an average efficiency improvement of 35.472% under a single thread. The efficiency can be further improved when more threads are enabled. The computational efficiency of proposed method under 4 threads is improved up to 214.313% compared to the original Octomap. In terms of memory consumption, the average memory requirement of the proposed method is only 42.437% of the original Octomap under a single thread, and the memory consumption only reaches a similar level of the original Octomap method when 3 threads are enabled.

7.2 Research Contributions

Compared with previous Octomap-based dynamic object detection methods, the method proposed in this paper has the following contributions:

First, the previous Octomap-based approaches usually require the generation of a very huge voxel grid, and thus are only applicable to sparse point clouds or require a large amount of computation and memory. The proposed method in this thesis segments the initial MLS point cloud into several data frames and inserts only two or three neighbor data frames into a single Octomap at a time, avoiding the generation of a huge Octomap by generating multiple smaller Octomaps, thus increasing efficiency and reducing memory requirements. Compared with the original Octomap method, the proposed method in this thesis is improved in terms of computational efficiency as well as memory saving.

Second, many previous studies defined non-ground space as the ROI for point cloud dynamic object detection tasks. Such ROI has only one explicitly defined lower boundary. Based on this, this thesis defines the local vehicle height restriction as the upper boundary of the ROI, further narrowing the ROI of dynamic object detection and extraction to reduce the computational effort in subsequent steps.

Third, the compatibility of the present method with parallel computing is demonstrated. A foundation is laid for further improving the efficiency of this method using some advanced parallel computing technologies in the future.

7.3 Future Work

Due to the limited research time, many of the new problems and ideas encountered during the study could not be further explored. Therefore, this section lists some directions for improvement and optimization of this thesis in the future.

Self-adaptive threshold values: The setting of the threshold values can have an impact on the detection result. In this research, operations such as free point extraction, noise removal, vegetation removal, and fake dynamic object filtering from Octomap require a series of reasonable thresholds to be set in advance. Taking the denoising operation as an example, if the threshold is set too large, many dynamic free points will be lost. If the threshold value is set too small, many noise points will remain in the result. Both above scenarios will affect the quality of the result. In the currently proposed method, all these thresholds are fixed. However, in the subsequent research, the MLS point cloud can be sampled first, and then the optimal threshold values for the current input data can be estimated self-adaptively based on the sampled points. With the introduction of adaptive thresholds, some expected changes in the environment can be handled automatically when the application scenario changes without extensive modifications to the method, thus improving the robustness of the proposed method.

Detection methods optimized for sparse dynamic objects: To solve the problem that some of the sparse dynamic objects in the implementation results cannot be detected completely, the future research can try to modify the parameters of the point-based spatial neighborhood query algorithm based on the distance from the MLS capture point to the sensor or the local point density of the object to achieve better detection results.

Extension to more application scenarios: The four case sites used in the implementation phase of this thesis are all located in several adjacent road sections in Delft, the Netherlands. So, there are many similarities in the overall environment among these four case sites. The performance of the proposed method can be further tested in the future in more environments, including highways and rural areas, to verify the generalizability of the method. Since no pedestrians are found in the case site data used in this study, the method needs to be subsequently validated for pedestrian detection in areas with high pedestrian traffic such as pedestrian streets, popular tourist spots, and commercial areas. In addition, the ROI in this thesis does not include high-altitude space, so it is also a worthwhile research to try whether this method can also be applied to the detection of moving objects such as birds, airplanes, and drones at high altitude.

Integration of better static object detection methods: Most mis-detected static objects are vegetation, so it is necessary to introduce more advanced vegetation detection algorithms to remove the remnant vegetation. For the ground extraction algorithm, local height changes need to be considered to obtain a more complete ground surface, especially when the terrain is not very flat. In addition, the well-performed pole-like object detection algorithm needs to be integrated into the workflow to exclude static objects such as streetlights, traffic lights, and traffic signs.

Speed detection and direction tracking: Octomap can only detect changes in the environment, but it cannot calculate the speed of dynamic objects and track the directions of them. But in many scenarios, such as autonomous driving, the moving direction of a dynamic object as well as its speed is also very important information. Therefore, the subsequent research can focus on how to obtain further motion information of dynamic objects after they are detected, including their direction and speed.

More realistic LiDAR ray simulation: The construction of Octomap is based on ray reconstruction. LiDAR rays are usually idealized to be a straight line in space. However, in the real environment the LiDAR ray may form a very small cone because of scattering and other factors. This means that the ray beam will end up hitting multiple voxel cells. Therefore, in future research, the intersection volume of the LiDAR ray and each hit voxel cell can be calculated first, and then the occupancy probability of this ray can be assigned to each hit voxel cell in proportion to the intersection volume.

Replacing the voxel grid with point-based structures: Using Octomap means introducing additional voxel grids in addition to the original MLS data, which increases the computational effort and memory consumption. Another problem is that in Octomap the probabilities of all points from one voxel cell are considered as the same by default. A follow-up idea for improvement is to replace the voxel grid structure in Octomap using a point-based structure such as an axis aligned bounding box tree (AABB tree)[1]. First construct a spherical or cube-shaped neighborhood for each point first, then calculate the number of rays hitting that neighborhood and directly calculate the probability of occupancy for each point. This improves the voxel cell-based occupancy probability to a point-based occupancy probability.

[1] https://doc.cgal.org/latest/AABB_tree/index.html

GPU-accelerated computation: For MLS data with a large number of points, using GPU-based parallel accelerated computing is more efficient than CPU-based computing. The method proposed in this thesis can be integrated with GPU technologies such as CUDA in subsequent research to achieve more efficient parallel computing.

Integration with DBMS: The Database Management Systems (DBMS) is well suited for managing and processing massive data like point clouds. The parallel operation of removing duplicate points that is not achieved in this thesis, but these duplicate points can be sorted and removed efficiently in the DBMS. The massive parallel computing technology in DBMS can improve the processing efficiency of point cloud data. So how to integrate this research with DBMS to further improve performance is also a topic well worth exploring in the future.

Bibliography

Anderson-Sprecher, P., Simmons, R., & Huber, D. (2011). Background subtraction and accessibility analysis in evidence grids. 2011 IEEE International Conference on Robotics and Automation, 3104–3110. https://doi.org/10.1109/ICRA.2011.5980428

Arisholm, G., Skauli, T., & Landrø, S. (2018). Combined range ambiguity resolution and noise reduction in lidar signal processing. Optical Engineering, 57(07), 1. https://doi.org/10.1117/1.OE.57.7.073103

Arora, M., Wiesmann, L., Chen, X., & Stachniss, C. (2021). Mapping the Static Parts of Dynamic Scenes from 3D LiDAR Point Clouds Exploiting Ground Segmentation. 2021 European Conference on Mobile Robots (ECMR), 1–6. https://doi.org/10.1109/ECMR50962.2021.9568799

Balado, J., Arias, P., Díaz-Vilariño, L., & González-deSantos, L. M. (2018). Automatic CORINE land cover classification from airborne LIDAR data. Procedia Computer Science, 126, 186–194. https://doi.org/10.1016/j.procs.2018.07.222

Balado, J., Díaz-Vilariño, L., Arias, P., & Lorenzo, H. (2019). Point clouds for direct pedestrian pathfinding in urban environments. ISPRS Journal of Photogrammetry and Remote Sensing, 148, 184–196. https://doi.org/10.1016/j.isprsjprs.2019.01.004

Balado, J., González, E., Verbree, E., Díaz-Vilariño, L., & Lorenzo, H. (2020). AUTOMATIC DETECTION AND CHARACTERIZATION OF GROUND OCCLUSIONS IN URBAN POINT CLOUDS FROM MOBILE LASER SCANNING DATA. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, VI-4/W1-2020, 13–20. https://doi.org/10.5194/isprs-annals-VI-4-W1-2020-13-2020

Banerjee, N., Lisin, D., Briggs, J., Llofriu, M., & Munich, M. E. (2019). Lifelong Mapping using Adaptive Local Maps. 2019 European Conference on Mobile Robots (ECMR), 1–8. https://doi.org/10.1109/ECMR.2019.8870347

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9), 509–517. https://doi.org/10.1145/361002.361007

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. IBM Systems Journal, 4(1), 25–30. https://doi.org/10.1147/sj.41.0025

Čerňava, J., Mokroš, M., Tuček, J., Antal, M., & Slatkovská, Z. (2019). Processing Chain for Estimation of Tree Diameter from GNSS-IMU-Based Mobile Laser Scanning Data. Remote Sensing, 11(6), 615. https://doi.org/10.3390/rs11060615

Charles, R. Q., Su, H., Kaichun, M., & Guibas, L. J. (2017). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 77–85. https://doi.org/10.1109/CVPR.2017.16

Che, E., Jung, J., & Olsen, M. (2019). Object Recognition, Segmentation, and Classification of Mobile Laser Scanning Point Clouds: A State of the Art Review. Sensors, 19(4), 810. https://doi.org/10.3390/s19040810

Chen, X., Milioto, A., Palazzolo, E., Giguère, P., Behley, J., & Stachniss, C. (2019). SuMa++: Efficient LiDAR-based Semantic SLAM. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4530–4537. https://doi.org/10.1109/IROS40897.2019.8967704

Chen, Y.-W., & Lin, C.-J. (2006). Combining SVMs with Various Feature Selection Strategies. In I. Guyon, M. Nikravesh, S. Gunn, & L. A. Zadeh (Eds.), Feature Extraction (Vol. 207, pp. 315–324). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-35488-8_13

Cheng, J., Xiang, Z., Cao, T., & Liu, J. (2014). Robust vehicle detection using 3D Lidar under complex urban environment. 2014 IEEE International Conference on Robotics and Automation (ICRA), 691–696. https://doi.org/10.1109/ICRA.2014.6906929

Choi, J., Ulbrich, S., Lichte, B., & Maurer, M. (2013). Multi-Target Tracking using a 3D-Lidar sensor for autonomous vehicles. 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), 881–886. https://doi.org/10.1109/ITSC.2013.6728343

Dalponte, M., Coops, N. C., Bruzzone, L., & Gianelle, D. (2009). Analysis on the Use of Multiple Returns LiDAR Data for the Estimation of Tree Stems Volume. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2(4), 310–318. https://doi.org/10.1109/JSTARS.2009.2037523

Dewan, A., Caselitz, T., Tipaldi, G. D., & Burgard, W. (2016). Motion-based detection and tracking in 3D LiDAR scans. 2016 IEEE International Conference on Robotics and Automation (ICRA), 4508–4513. https://doi.org/10.1109/ICRA.2016.7487649

Di Stefano, F., Chiappini, S., Gorreja, A., Balestra, M., & Pierdicca, R. (2021). Mobile 3D scan LiDAR: A literature review. Geomatics, Natural Hazards and Risk, 12(1), 2387–2429. https://doi.org/10.1080/19475705.2021.1964617

Ding, P., & Wang, Z. (2021). 3D LiDAR Point Cloud Loop Detection Based on Dynamic Object Removal. 2021 IEEE International Conference on Real-Time Computing and Robotics (RCAR), 980–985. https://doi.org/10.1109/RCAR52367.2021.9517428

Endo, Y., Javanmardi, E., & Kamijo, S. (2021). Analysis of Occlusion Effects for Map-Based Self-Localization in Urban Areas. Sensors, 21(15), 5196. https://doi.org/10.3390/s21155196

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381–395. https://doi.org/10.1145/358669.358692

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research, 32(11), 1231–1237. https://doi.org/10.1177/0278364913491297

Gonzalez, R. C., & Woods, R. E. (2008). Digital image processing (3rd ed.). Prentice Hall.

Gorte, B. G. H. (2002). Segmentation of tin-structured surface models. Joint International Symposium on Geospatial Theory, Processing and Applications, 1–5. https://www.isprs.org/proceedings/xxxiv/part4/pdfpapers/351.pdf (accessed April 20, 2022)

Guo, Q., Su, Y., Hu, T., Guan, H., Jin, S., Zhang, J., Zhao, X., Xu, K., Wei, D., Kelly, M., & Coops, N. C. (2021). Lidar Boosts 3D Ecological Observations and Modelings: A Review and Perspective. IEEE Geoscience and Remote Sensing Magazine, 9(1), 232–257. https://doi.org/10.1109/MGRS.2020.3032713

Guo, Z., Cai, B., Jiang, W., & Wang, J. (2019). Feature-based detection and classification of moving objects using LiDAR sensor. IET Intelligent Transport Systems, 13(7), 1088–1096. https://doi.org/10.1049/iet-its.2018.5291

Gupta, A., Byrne, J., Moloney, D., Watson, S., & Yin, H. (2020). Tree Annotations in LiDAR Data Using Point Densities and Convolutional Neural Networks. IEEE Transactions on Geoscience and Remote Sensing, 58(2), 971–981. https://doi.org/10.1109/TGRS.2019.2942201

Hauser, D., Glennie, C., & Brooks, B. (2016). Calibration and Accuracy Analysis of a Low-Cost Mapping-Grade Mobile Laser Scanning System. Journal of Surveying Engineering, 142(4), 04016011. https://doi.org/10.1061/(ASCE)SU.1943-5428.0000178

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots, 34(3), 189–206. https://doi.org/10.1007/s10514-012-9321-0

Hu, X., Li, X., & Zhang, Y. (2013). Fast Filtering of LiDAR Point Cloud in Urban Areas Based on Scan Line Segmentation and GPU Acceleration. IEEE Geoscience and Remote Sensing Letters, 10(2), 308–312. https://doi.org/10.1109/LGRS.2012.2205130

Huang, R., Zhang, W., Kundu, A., Pantofaru, C., Ross, D. A., Funkhouser, T., & Fathi, A. (2020). An LSTM Approach to Temporal 3D Object Detection in LiDAR Point Clouds. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), Computer Vision – ECCV 2020 (pp. 266–282). Springer International Publishing. https://doi.org/10.1007/978-3-030-58523-5_16

Hyyppä, J., Jaakkola, A., Chen, Y., Kukko, A., Kaartinen, H., Zhu, L., Alho, P., & Hyyppä, H. (2013). Unconventional LIDAR mapping from air, terrestrial and mobile. Proceedings of the Photogrammetric Week, 205–214. https://ifpwww.ifp.uni-stuttgart.de/publications/phowo13/180Hyyppae.pdf (Accessed April 20, 2022)

Iqbal, H., Campo, D., Marin-Plaza, P., Marcenaro, L., Gómez, D. M., & Regazzoni, C. (2021). Modeling Perception in Autonomous Vehicles via 3D Convolutional Representations on LiDAR. IEEE Transactions on Intelligent Transportation Systems, 1–12. https://doi.org/10.1109/TITS.2021.3130974

Kampffmeyer, M., Dong, N., Liang, X., Zhang, Y., & Xing, E. P. (2019). ConnNet: A Long-Range Relation-Aware Pixel-Connectivity Network for Salient Segmentation. IEEE Transactions on Image Processing, 28(5), 2518–2529. https://doi.org/10.1109/TIP.2018.2886997

Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., Happold, M., Herman, H., Mandelbaum, R., Pilarski, T., Rander, P., Thayer, S., Vallidis, N., & Warner, R. (2006). Toward Reliable Off Road Autonomous Vehicles Operating in Challenging Environments. The International Journal of Robotics Research, 25(5–6), 449–483. https://doi.org/10.1177/0278364906065543

Kim, G., & Kim, A. (2020). Remove, then Revert: Static Point cloud Map Construction using Multiresolution Range Images. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 10758–10765. https://doi.org/10.1109/IROS45743.2020.9340856

Kiran, B. R., Roldão, L., Irastorza, B., Verastegui, R., Süss, S., Yogamani, S., Talpaert, V., Lepoutre, A., & Trehard, G. (2019). Real-Time Dynamic Object Detection for Autonomous Driving Using Prior 3D-Maps. In L. Leal-Taixé & S. Roth (Eds.), Computer Vision – ECCV 2018 Workshops (Vol. 11133, pp. 567–582). Springer International Publishing. https://doi.org/10.1007/978-3-030-11021-5_35

Ku, J., Mozifian, M., Lee, J., Harakeh, A., & Waslander, S. L. (2018). Joint 3D Proposal Generation and Object Detection from View Aggregation. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1–8. https://doi.org/10.1109/IROS.2018.8594049

Lim, H., Hwang, S., & Myung, H. (2021). ERASOR: Egocentric Ratio of Pseudo Occupancy-Based Dynamic Object Removal for Static 3D Point Cloud Map Building. IEEE Robotics and Automation Letters, 6(2), 2272–2279. https://doi.org/10.1109/LRA.2021.3061363

Lin, Z., Hashimoto, M., Takigawa, K., & Takahashi, K. (2018). Vehicle and Pedestrian Recognition Using Multilayer Lidar based on Support Vector Machine. 2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), 1–6. https://doi.org/10.1109/M2VIP.2018.8600877

Luo, W., Yang, B., & Urtasun, R. (2018). Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 3569–3577. https://doi.org/10.1109/CVPR.2018.00376

Ma, L., Li, Y., Li, J., Wang, C., Wang, R., & Chapman, M. A. (2018). Mobile Laser Scanned Point-Clouds for Road Object Detection and Extraction: A Review. Remote Sensing, 10(10), 1531. https://doi.org/10.3390/rs10101531

Ma, W.-C., Urtasun, R., Tartavull, I., Barsan, I. A., Wang, S., Bai, M., Mattyus, G., Homayounfar, N., Lakshmikanth, S. K., & Pokrovsky, A. (2019). Exploiting Sparse Semantic HD Maps for Self-Driving Vehicle Localization. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 5304–5311. https://doi.org/10.1109/IROS40897.2019.8968122

Maas, H.-G., & Vosselman, G. (1999). Two algorithms for extracting building models from raw laser altimetry data. ISPRS Journal of Photogrammetry and Remote Sensing, 54(2), 153–163. https://doi.org/10.1016/S0924-2716(99)00004-0

Mekala, M. S., Park, W., Dhiman, G., Srivastava, G., Park, J. H., & Jung, H.-Y. (2021). Deep Learning Inspired Object Consolidation Approaches Using LiDAR Data for Autonomous Driving: A Review. Archives of Computational Methods in Engineering. https://doi.org/10.1007/s11831-021-09670-y

Milioto, A., Vizzo, I., Behley, J., & Stachniss, C. (2019). RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4213–4220. https://doi.org/10.1109/IROS40897.2019.8967762

Moosmann, F., Pink, O., & Stiller, C. (2009). Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion. 2009 IEEE Intelligent Vehicles Symposium, 215–220. https://doi.org/10.1109/IVS.2009.5164280

Moravec, H., & Elfes, A. (1985). High resolution maps from wide angle sonar. 1985 IEEE International Conference on Robotics and Automation Proceedings, 2, 116–121. https://doi.org/10.1109/ROBOT.1985.1087316

Najdataei, H., Nikolakopoulos, Y., Gulisano, V., & Papatriantafilou, M. (2018). Continuous and Parallel LiDAR Point-Cloud Clustering. 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 671–684. https://doi.org/10.1109/ICDCS.2018.00071

Okyay, U., Telling, J., Glennie, C. L., & Dietrich, W. E. (2019). Airborne lidar change detection: An overview of Earth sciences applications. Earth-Science Reviews, 198, 102929. https://doi.org/10.1016/j.earscirev.2019.102929

Oršulić, J., Milijas, R., Batinovic, A., Markovic, L., Ivanovic, A., & Bogdan, S. (2021). Flying with Cartographer: Adapting the Cartographer 3D Graph SLAM Stack for UAV Navigation. 2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO), 1–7. https://doi.org/10.1109/AIRPHARO52252.2021.9571065

Otepka, J., Ghuffar, S., Waldhauser, C., Hochreiter, R., & Pfeifer, N. (2013). Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. ISPRS International Journal of Geo-Information, 2(4), 1038–1065. https://doi.org/10.3390/ijgi2041038

Otepka, J., Mandlburger, G., Karel, W., Wöhrer, B., Ressl, C., & Pfeifer, N. (2021). A FRAMEWORK FOR GENERIC SPATIAL SEARCH IN 3D POINT CLOUDS. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, V-2–2021, 35–42. https://doi.org/10.5194/isprs-annals-V-2-2021-35-2021

Pagad, S., Agarwal, D., Narayanan, S., Rangan, K., Kim, H., & Yalla, G. (2020). Robust Method for Removing Dynamic Objects from Point Clouds. 2020 IEEE International Conference on Robotics and Automation (ICRA), 10765–10771. https://doi.org/10.1109/ICRA40945.2020.9197168

Petrovskaya, A., & Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. Autonomous Robots, 26(2), 123–139. https://doi.org/10.1007/s10514-009-9115-1

Pfeifer, N., Falkner, J., Bayr, A., Eysn, L., & Ressl, C. (2021). Test Charts for Evaluating Imaging and Point Cloud Quality of Mobile Mapping Systems for Urban Street Space Acquisition. Remote Sensing, 13(2), 237. https://doi.org/10.3390/rs13020237

Pfreundschuh, P., Hendrikx, H. F. C., Reijgwart, V., Dubé, R., Siegwart, R., & Cramariuc, A. (2021). Dynamic Object Aware LiDAR SLAM based on Automatic Generation of Training Data. 2021 IEEE International Conference on Robotics and Automation (ICRA), 11641 – 11647. https://doi.org/10.1109/ICRA48506.2021.9560730

Pomerleau, F., Krüsi, P., Colas, F., Furgale, P., & Siegwart, R. (2014). Long-term 3D map maintenance in dynamic environments. 2014 IEEE International Conference on Robotics and Automation (ICRA), 3712–3719. https://doi.org/10.1109/ICRA.2014.6907397

Postica, G., Romanoni, A., & Matteucci, M. (2016). Robust moving objects detection in lidar data exploiting visual cues. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1093–1098. https://doi.org/10.1109/IROS.2016.7759185

Rodríguez-Cuenca, B., García-Cortés, S., Ordóñez, C., & Alonso, M. C. (2015). An approach to detect and delineate street curbs from MLS 3D point cloud data. Automation in Construction, 51, 103–112. https://doi.org/10.1016/j.autcon.2014.12.009

Sánchez-Cruz, H., Sossa-Azuela, H., Braumann, U.-D., & Bribiesca, E. (2013). The Euler-Poincaré Formula Through Contact Surfaces of Voxelized Objects. Journal of Applied Research and Technology, 11(1), 65–78. https://doi.org/10.1016/S1665-6423(13)71515-3

Schauer, J., & Nüchter, A. (2018). The Peopleremover—Removing Dynamic Objects From 3-D Point Cloud Data by Traversing a Voxel Occupancy Grid. IEEE Robotics and Automation Letters, 3(3), 1679–1686. https://doi.org/10.1109/LRA.2018.2801797

Shackleton, J., VanVoorst, B., & Hesch, J. (2010). Tracking People with a 360-Degree Lidar. 2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance, 420–426. https://doi.org/10.1109/AVSS.2010.52

Shi, S., Wang, X., & Li, H. (2019). PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 770–779. https://doi.org/10.1109/CVPR.2019.00086

Soilán, Sánchez-Rodríguez, Río-Barral, Perez-Collazo, Arias, & Riveiro. (2019). Review of Laser Scanning Technologies and Their Applications for Road and Railway Infrastructure Monitoring. Infrastructures, 4(4), 58. https://doi.org/10.3390/infrastructures4040058

Sugumaran, R., Oryspayev, D., & Gray, P. (2011). GPU-based cloud performance for LiDAR data processing. Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications - COM.Geo '11, 1. https://doi.org/10.1145/1999320.1999369

Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., … Anguelov, D. (2020). Scalability in Perception for Autonomous Driving: Waymo Open Dataset. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2443–2451. https://doi.org/10.1109/CVPR42600.2020.00252

Teo, T.-A., & Shih, T.-Y. (2013). Lidar-based change detection and change-type determination in urban areas. International Journal of Remote Sensing, 34(3), 968–981. https://doi.org/10.1080/01431161.2012.714504

Tombari, F., Salti, S., & Di Stefano, L. (2010). Unique signatures of histograms for local surface description. Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III, 356–369. https://doi.org/10.1007/978-3-642-15558-1_26

Ushani, A. K., Wolcott, R. W., Walls, J. M., & Eustice, R. M. (2017). A learning approach for real-time temporal scene flow estimation from LIDAR data. 2017 IEEE International Conference on Robotics and Automation (ICRA), 5666–5673. https://doi.org/10.1109/ICRA.2017.7989666

Ussyshkin, V., & Theriault, L. (2011). Airborne Lidar: Advances in Discrete Return Technology for 3D Vegetation Mapping. Remote Sensing, 3(3), 416–434. https://doi.org/10.3390/rs3030416

Wang, C., Meng, L., She, S., Mitchell, I. M., Li, T., Tung, F., Wan, W., Meng, Max. Q.-H., & de Silva, C. W. (2017). Autonomous mobile robot navigation in uneven and unstructured indoor environments. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 109–116. https://doi.org/10.1109/IROS.2017.8202145

Wang, Y., Chen, Q., Zhu, Q., Liu, L., Li, C., & Zheng, D. (2019). A Survey of Mobile Laser Scanning Applications and Key Techniques over Urban Areas. Remote Sensing, 11(13), 1540. https://doi.org/10.3390/rs11131540

Weinmann, M., Jutzi, B., Hinz, S., & Mallet, C. (2015). Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. ISPRS Journal of Photogrammetry and Remote Sensing, 105, 286–304. https://doi.org/10.1016/j.isprsjprs.2015.01.016

Wen, W. W., Zhang, G., & Hsu, L.-T. (2021). GNSS NLOS Exclusion Based on Dynamic Object Detection Using LiDAR Point Cloud. IEEE Transactions on Intelligent Transportation Systems, 22(2), 853–862. https://doi.org/10.1109/TITS.2019.2961128

Williams, K., Olsen, M. J., Roe, G. V., & Glennie, C. (2013). Synthesis of Transportation Applications of Mobile LIDAR. Remote Sensing, 5(9), 4652–4692. https://doi.org/10.3390/rs5094652

Wu, Y., Wang, Y., Zhang, S., & Ogai, H. (2021). Deep 3D Object Detection Networks Using LiDAR Data: A Review. IEEE Sensors Journal, 21(2), 1152–1171. https://doi.org/10.1109/JSEN.2020.3020626

Xia, Y., Sun, Z., Tok, A., & Ritchie, S. (2022). A dense background representation method for traffic surveillance based on roadside LiDAR. Optics and Lasers in Engineering, 152, 106982. https://doi.org/10.1016/j.optlaseng.2022.106982

Xiao, W., Vallet, B., Schindler, K., & Paparoditis, N. (2016). Street-side vehicle detection, classification and change detection using mobile laser scanning data. ISPRS Journal of Photogrammetry and Remote Sensing, 114, 166–178. https://doi.org/10.1016/j.isprsjprs.2016.02.007

Xu, S., Cheng, P., Zhang, Y., & Ding, P. (2015). Error Analysis and Accuracy Assessment of Mobile Laser Scanning System. The Open Automation and Control Systems Journal, 7(1), 485–495. https://doi.org/10.2174/1874444301507010485

Xu, S., Oude Elberink, S., & Vosselman, G. (2012). ENTITIES AND FEATURES FOR CLASSIFCATION OF AIRBORNE LASER SCANNING DATA IN URBAN AREA. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, I–4, 257–262. https://doi.org/10.5194/isprsannals-I-4-257-2012

Yao, W., Hinz, S., & Stilla, U. (2010). Airborne analysis and assessment of urban traffic scenes from LiDAR data—Theory and experiments. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, 75–82. https://doi.org/10.1109/CVPRW.2010.5543901

Yao, W., Hinz, S., & Stilla, U. (2011). Extraction and motion estimation of vehicles in single-pass airborne LiDAR data towards urban traffic analysis. ISPRS Journal of Photogrammetry and Remote Sensing, 66(3), 260–271. https://doi.org/10.1016/j.isprsjprs.2010.10.005

Yin, H., Wang, Y., Ding, X., Tang, L., Huang, S., & Xiong, R. (2020). 3D LiDAR-Based Global Localization Using Siamese Neural Network. IEEE Transactions on Intelligent Transportation Systems, 21(4), 1380–1392. https://doi.org/10.1109/TITS.2019.2905046

Yoon, D., Tang, T., & Barfoot, T. (2019). Mapless Online Detection of Dynamic Objects in 3D Lidar. 2019 16th Conference on Computer and Robot Vision (CRV), 113–120. https://doi.org/10.1109/CRV.2019.00023

Zeeshan Zia, M., Stark, M., Schiele, B., & Schindler, K. (2013). Detailed 3D Representations for Object Recognition and Modeling. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(11), 2608–2623. https://doi.org/10.1109/TPAMI.2013.87

Zhang, L., Li, Q., Li, M., Mao, Q., & Nüchter, A. (2013). Multiple Vehicle-like Target Tracking Based on the Velodyne LiDAR*. IFAC Proceedings Volumes, 46(10), 126–131. https://doi.org/10.3182/20130626-3-AU-2035.00058

Zhang, T., & Jin, P. J. (2022). Roadside Lidar Vehicle Detection and Tracking Using Range And Intensity Background Subtraction. ArXiv:2201.04756 [Cs, Eess]. https://doi.org/10.48550/arXiv.2201.04756

Zhao, L., & Thorpe, C. (1998). Qualitative and quantitative car tracking from a range image sequence. Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231), 496–501. https://doi.org/10.1109/CVPR.1998.698651

Zhou, Y., & Tuzel, O. (2018). VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4490–4499. https://doi.org/10.1109/CVPR.2018.00472