TRR
JOURNAL OF THE TRANSPORTATION RESEARCH BOARD

Research Article

Transportation Research Record
2022, Vol. 2676(6) 315–329

DOI: 10.1177/03611981221074365
journals.sagepub.com/home/trr
SAGE

Fast-Spherical-Projection-Based Point Cloud Clustering Algorithm

Zhihui Chen[1], Hao Xu[1], Junxuan Zhao[2], and Hongchao Liu[2]

Abstract

Roadside LiDAR (light detection and ranging) is a solution to fill in the gaps for connected vehicles (CV) by detecting the status of global road users at transportation facilities. It relies greatly on the clustering algorithm for accurate and rapid data processing so as to ensure effectiveness and reliability. To contribute to better roadside LiDAR-based transportation facilities, this paper presents a fast-spherical-projection-based clustering algorithm (FSPC) for real-time LiDAR data processing with higher clustering accuracy and noise handling. The FSPC is designed to work on a spherical map which could be directly derived from the instant returns of a LiDAR sensor. A 2D-window searching strategy is specifically designed to accelerate the computation and alleviate the density variation impact in the LiDAR point cloud. The test results show the proposed algorithm can achieve a high processing efficiency with 24.4 ms per frame, satisfying the real-time requirement for most common LiDAR applications (100 ms per frame), and it also ensures a high accuracy in object clustering, with 96%. Additionally, it is observed that the proposed FSPC allows a wider detection range and is more stable, tackling the surge in foreground points that frequently occurs in roadside LiDAR applications. Finally, the generality of the proposed FSPC indicates the proposed algorithm could also be implemented in other areas such as autonomous driving and remote sensing.

Keywords

data and data science, automatic vehicle detection and identification systems, computer vision, data science, remote sensing, vehicle detection

Connected-vehicle (CV) technology is an emerging technology to reduce crashes and increase energy efficiency in the transportation system, one which enables bi-directional communications between infrastructure and connected road users to share real-time information and provide rapid response to potential events and operation enhancement (*1*, *2*). However, the currently deployed CV system is not performing as well as expected. The primary reason is the information gap resulting from unconnected vehicles, pedestrians, bicycles, or wild animals (*3*).

Roadside LiDAR is one of the solutions that supply complementary information by using infrastructure-based LiDAR (light detection and ranging) sensors to track the real-time status of those unconnected road users within detection range. The LiDAR sensor is an advanced technology, returning accurate and dense point clouds from the surroundings, which has been solidly applied to autonomous driving for its great robustness against different light conditions and high precision on 3D point cloud collection (*3*). For an infrastructure-based roadside LiDAR, the most crucial step is to detect the target objects in the environment, and its efficiency is important since a faster detection speed will supply more time for the post-detection response to the road users in the real world. On the other hand, detection accuracy is also a critical factor in ensuring the reliability of the LiDAR-based system. Similar to the image processing pipeline, LiDAR-based detection is also needed to extract the foreground points and finely partition those points as different entities (clusters).

[1]Department of Civil & Environmental Engineering, University of Nevada, Reno, Reno, NV
[2]Department of Civil, Environmental and Construction Engineering, Texas Tech University, Lubbock, TX

Corresponding Author:
Junxuan Zhao, junxuan09@gmail.com

Existing point cloud clustering algorithms could be coarsely categorized as two types according to the format of input data: (1) direct cluster point clouds with an array of point coordinates (in 3D or 2D space); (2) pixel-based clustering.

The first type of clustering is intuitive, whereby the input for the clustering algorithm is an array of 2D or 3D point coordinates. For the scenario of LiDAR-based clustering, the number of target objects in the scenes is impossible to predict, which limits the applicable algorithms. Those clustering algorithms that need to assign the number of clusters, such as K-means, GMM, and so forth, are inapplicable. The most popular algorithm for this type of clustering task is DBSCAN (density-based spatial clustering of applications with noise). Two density-related parameters are involved in the DBSCAN: the **minPts** (minimum points for a cluster) and the **eps** (the radius to search neighborhoods). Using these, the DBSCAN can detect the arbitrary shape of clusters based on point density, and the points with low density are treated as noise. These two features are quite useful in LiDAR point cloud clustering (*4*). However, the most challenging part for the DBSCAN is the variant density of the LiDAR point cloud. Generally speaking, the variant density results from the radiation property of laser beams. For example, given two adjacent laser beams, the Euclidean distance between two returned points would vary according to the distance from the points to the LiDAR sensor (a longer distance would contribute to a larger interval between adjacent points). In other words, the parameters (minPts and eps) to identify the same object are different given different object distances. This problem will greatly affect the result of DBSCAN because the variant point density is difficult to capture by a fixed set of parameters. To date, many pieces of work have been carried out to solve this problem. Zhao et al. (*5*) and Chen et al. (*6*), explicitly modeled the target objects using geometric models to figure out the point distribution features such as the theoretical point number given a different detection range. Zhao et al. (*5*) modified the DBSCAN, applying a set of dynamic DBSCAN parameters on different detection ranges to improve the accuracy of clustering. Wang et al. (*7*) proposed an automatic parameter estimation method to extract the intrinsic pattern in the point cloud, by which the target objects in different scenes are maximally detected. Similar work may be found in Zhao et al. (*8*), where the algorithm parameters are estimated by a Gaussian map. In sum, the traditional DBSCAN suffers from the varying point density resulting from the LiDAR sensor itself. The DBSCAN variants attempt to modify the algorithm by adjusting the parameter selection strategy. Even though the accuracy problem could be solved, the time complexity remains a problem. Generally, the time complexity for traditional DBSCAN and its variants is $O(N^2)$. (If some tree-based data structures were to be applied, the time complexity could be improved to $O(N\log(N))$.). The exponentially increasing (or log trend) time consumption would heavily impair the real-time property of roadside LiDAR applications such as the periodic surge of foreground points caused by traffic peaks. Moreover, the point number for LiDAR is getting larger with the development of LiDAR technology, and it is difficult to achieve real-time processing speed using traditional approaches. To date, the maximum point number returned by Velodyne 128-LiDAR would come to 2.6 million points per second (*9*) which is two to four times greater than current widely adopted LiDAR devices, and existing algorithms have difficulty satisfying this quantitative level.

The other form of point cloud clustering first converts the 3D point cloud into a 2D array and further utilizes image processing to accelerate the clustering. Two types of projection are usually applied: (1) XY-plane projection and (2)spherical projection. The XY-plane projection is intuitive: it first establishes an XY-grid, then projects 3D coordinates onto the established grid (*10*–*14*). The projected grid is essentially an occupancy array with a binary value indicating the presence of a point on each grid cell. After that, the clustering is implemented on the projected grid, grouping the presence of "pixels" into clusters. Even though it has reduced the dimension of points and increased the efficiency, it may lose the detailed information in the 3D point cloud, which is harmful to further steps of LiDAR data processing such as the classification task. In recent studies, another type of projection converting 3D points into a 2D array according to the spherical information is applied in some works (*15*, *16*). This type of projection (the spherical map) is related to the structure of the LiDAR sensor, which treats the azimuth as column index and laser channel as row index and projects the corresponding distance value in a 2D array. The spherical map could be processed by mature 2D image processing algorithms and remain informative. As for the image processing algorithm, existing works usually borrow the idea from the algorithms based on CCL (connected-component labeling). Most typical CCL-based algorithms consist of two steps (*15*, *17*, *18*): (1) assign provisional labels to each foreground pixel, and (2) create an equivalent scheme to integrate different assigned labels to each object, and use the different unique representative labels to represent each object. The equivalent scheme is created by the interconnected relationship among pixels. Two pixels are considered in the same cluster as they are directly or indirectly connected.

For the binary image (occupancy grid), CCL algorithms could be directly implemented. Himmelsbach et al. (*10*) first calculated an XY-plane occupancy grid from the 3D point cloud. After that, the standard CCL algorithm is applied to identify different objects in the 2D grid. Similar to Himmelsbach et al. (*10*), Börcs et al. (*14*) proposed a hierarchical grid structure with a macroscale grid to locate the object and a sub-grid for storing the detail point cloud. Standard CCL is adopted to cluster the macroscale grid. Other works also applying the binary grid associated with the CCL algorithm can be found in Himmelsbach et al. (*10*) and Börcs et al. (*14*).

Unlike the occupancy grid, the standard CCL algorithms are incompatible because the pixel-wise connections cannot be directly inferred from the projected distance value. The CCL algorithm could be modified, however, to accommodate the spherical map. Bogoslavskyi and Stachniss (*16*) proposed a modified CCL algorithm to cluster the spherical map by exploiting the angle between the laser beam and the line established by adjacent points. It is more possible to separate two points into two object clusters as the angle decreases. The clusters are identified by scanning the connectivity between "pixels" on the spherical map, and the connectivity is justified by defining an angle threshold. Inspired by Bogoslavskyi and Stachniss (*16*), Hasecke et al. (*19*) further proposed a Euclidean distance threshold to connect pixels. The Euclidean distance between pixels is calculated from the cosine law. The Euclidean threshold can be determined by the physical limitation between two objects, so the result can be more reasonable. Zermas et al. (*15*) proposed a multi-layer data structure, the idea of which is conceptually identical to the spherical map. A two-run CCL is adopted to cluster the points in the spherical map by merging adjacent points given a defined distance threshold.

In summary, existing LiDAR point cloud clustering algorithms focus on DBSCAN-based algorithms (for spatial clustering) and CCL-based algorithms (for pixel-wise clustering). The DBSCAN-based algorithms are effective in point clustering through their density-based cluster detection, while the efficiency is not ideal in real-time applications. On the other hand, the CCL-based algorithms are efficient because of their low time complexity. However, the CCL-based algorithms would output noise as the detected object (false-positive cases), since they cannot identify noise in the point cloud. This indicates that the CCL-based algorithm should be applied without noise involved in the scene, which is tough for many large scenes such as in Wu et al. (*3*) and Zhao et al. (*5*). To further exclude the noise in the output, the CCL-based algorithms require extra steps. Second, the CCL-based algorithms require high connectivities between adjacent pixels (points) to identify the connected components. For the roadside LiDAR, long-range detection may contribute to a high frequency of package loss; therefore, compared with DBSCAN, the CCL-based algorithms more easily fail in the roadside LiDAR applications because of the disconnections between points.

Concerning both computational load and accuracy of the clustering process, this work proposes a clustering algorithm known as FSPC (fast-spherical-projection-based clustering algorithm) that accelerates the clustering process and ensures the clustering accuracy as well. The proposed FSPC benefits from the pixel-wise spherical map data structure enabling a higher searching efficiency. Also, it utilizes the robust density-based clustering mainstream to accurately cluster objects with the functionality of noise exclusion. Meanwhile, it is proved that the proposed FSPC could be effective against the density variation problem in the LiDAR point cloud. The proposed algorithm is tuned and tested on a labeled data set collected at an intersection. The following article is organized as follows: methodology is explicitly presented in the section entitled Methods; the experiments covering tuning and performance testing are presented in the Results section; the major findings are given in the Conclusions.

Methods

In this section, the details of the proposed FSPC algorithm are presented. The proposed FSPC has a similar mainstream to that of traditional DBSCAN but has different implementation details. The major differences are twofold: (1) the input for FSPC is a spherical projected 2D map (spherical map), while the input for DBSCAN is an array of coordinates; (2) the neighborhood query of DBSCAN is achieved by an eps-radius spatial search, while the FSPC searches the neighborhood using a 2D-window on the spherical map. It should be noted that this work only focuses on the clustering algorithm; the input is the post-background-filtering point clouds with scattered noise points.

Spherical Map Representation

The input for FSPC is a 2D spherical map instead of a set of 3D Cartesian coordinates, which can be obtained directly from LiDAR returned packets. The LiDAR sensor collects the point cloud by rotationally firing a set of vertically arranged laser beams at a constant spinning rate. Each laser beam (a laser channel) is fixed at a particular elevation. If a laser beam hits an obstacle, the LiDAR sensor will report the relative distance from the LiDAR sensor to the surface of the obstacle. Figure 1 presents the top and side view of a LiDAR sensor and an

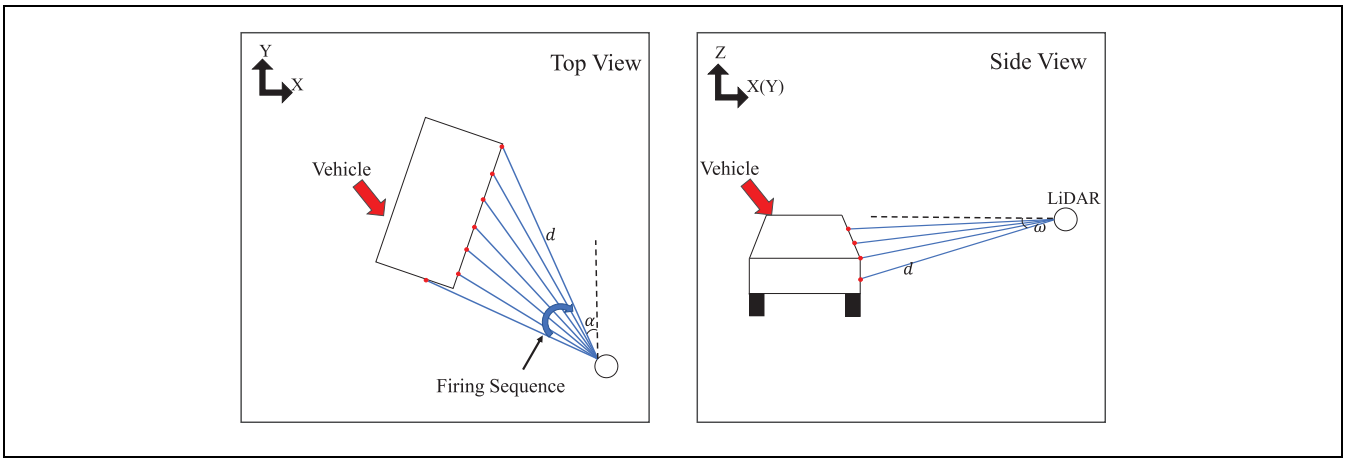

Figure 1. Example of azimuth and elevation angle.

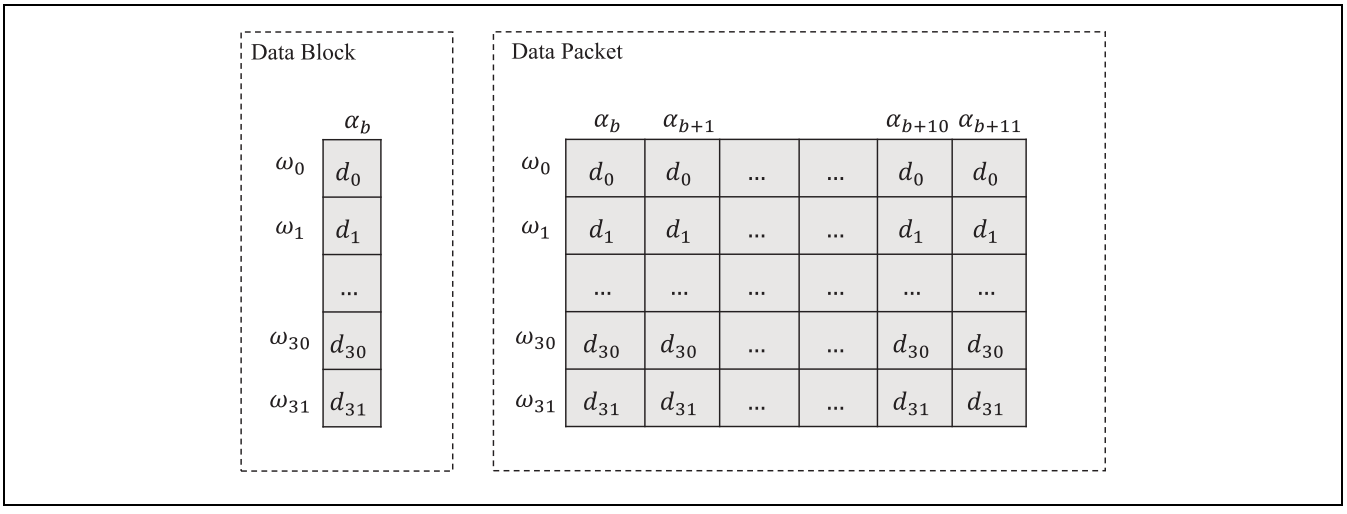

Figure 2. Example of data block and data packet.

outline of the vehicle. Multiple laser beams hit the surface of the vehicle and result in the presence of points represented by red dots. Three parameters describing the spherical data are returned at each firing: d, α, ω. Parameter d is the relative 3D distance from the laser hitting surface to the LiDAR sensor. The α is the azimuth angle (the horizontal angle on the XY plane) and the ω is the elevation angle relative to the XY plane.

Taking the Velodyne LiDAR sensor as an example, a data packet is generated after every 12 firings, and each firing contains a data block. Each data block includes 16–128 data points, depending on the number of laser channels. An example of the data block and packet for a 32-laser LiDAR sensor is given in Figure 2. α_b refers to a returned azimuth angle indicating the heading direction of a series of laser beams at the moment of firing, and ω_0 to ω_{31} correspondingly refer to the fixed elevation angles for each laser channel. A data block could be treated as a column for a data packet, and each data packet includes 12 returned blocks. The spinning direction of laser beams complies with the continuously increased azimuth angle from α_b to α_{b+11}, which also follows a counterclockwise firing sequence. For the default LiDAR setting, a complete 360° rotation (a frame of data) would take 0.1 s and contain 1800 firings. Thus, the azimuth difference between adjacent blocks (azimuth resolution) should be 0.2° (360/1800).

A spherical map is established after the LiDAR sensor completes a 360° spin. The spherical map in this study is defined as an array (shown in Figure 3) in which columns refer to different discretized azimuth channels and each row refers to the different laser channels. Each cell for the spherical map records the returned distance value according to the corresponding azimuth and laser

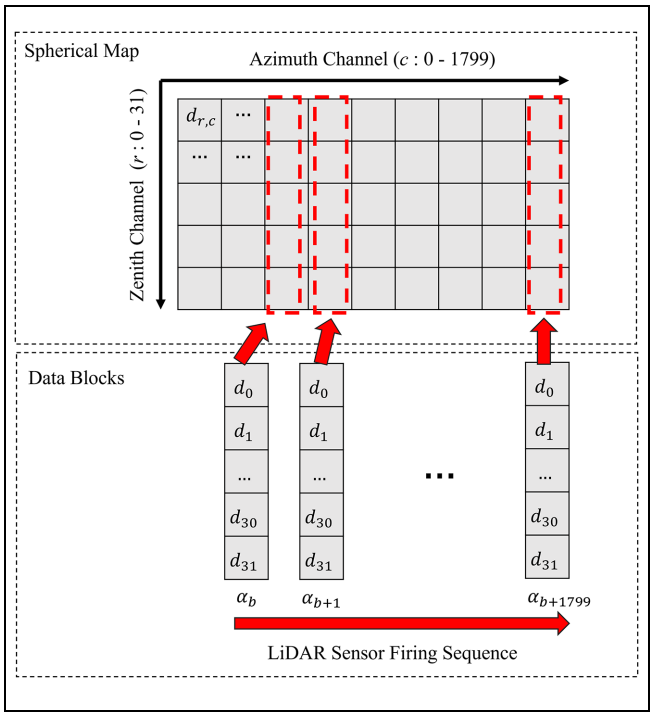

Figure 3. Example for spherical map and establishment.

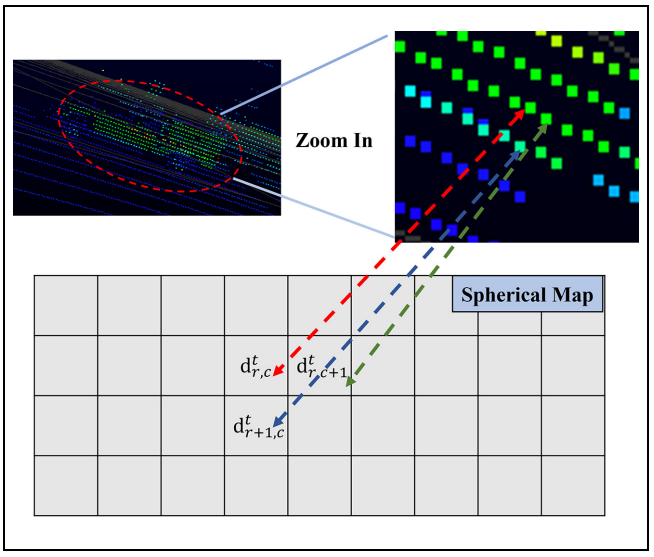

Figure 4. Point distribution in 3D space and on spherical map.
Note: DBSCAN = density-based spatial clustering of applications with noise; FSPC = fast spherical projection clustering.

channel. The process of spherical map establishment is also given in Figure 3. At beginning of each frame, an empty array with 1800×32 is created as a container of a spherical map. When a data block (a column of data array with 1×32) is transmitted from the LiDAR sensor, the azimuth value is first discretized as the azimuth channel (for 0–1799), which is also regarded as the column index of the spherical map. After that, the obtained data block is written into the container of the spherical map according to discretized azimuth channel azimuth. Once a roll-over (a complete 360°) of azimuth is detected, the container is output as the spherical map of the current frame.

FSPC Algorithm

The mainstream of FSPC is basically identical to the traditional DBSCAN, while the detailed implementations are different. The DBSCAN is one of the most prevalent algorithms in the clustering task; it can identify the arbitrary shape of clusters based on the spatial density while ruling out the noise. In general, two parameters are involved in the DBSCAN: **minPts** (minimum sample points to establish a core point) and **eps** (neighborhood searching radius). The two parameters work together to identify the dense point clusters.

The procedure of DBSCAN is briefly introduced as follows. (1) For each input point, search the neighborhoods within the eps-radius, and mark those points whose number of neighborhoods within eps-radius satisfies the minPts threshold for core points. (2) Identify the connected components of core points based on the direct and indirect interconnection of neighborhoods, and ignore the non-core points. (3) Assign each non-core point to a nearby connected component if the non-core point is within the eps-radius; otherwise, the non-core points are labeled as noise.

As suggested in the DBSCAN, the time complexity is composed of two parts: (1) the time for identifying the core points, which takes $O(N)$ time complexity; and (2) the time for searching the neighborhoods, which takes $O(N)$ per point at worst. If a data structure such as KD-tree is applied to improve the neighborhood searching process, the time complexity will be improved to $O(\log(N))$. In summary, the total time complexity is from $O(N^2)$ to $O(N\log(N))$. For those heavy traffic conditions that involve many moving objects, the large number of foreground points would exponentially increase the time consumption for clustering, and further result in lag in the real-time LiDAR application.

In the FSPC, the neighborhood searching is modified to improve the time complexity. In contrast to DBSCAN, the FSPC searches the neighborhoods on a spherical map. Figure 4 presents the point arrangement of a vehicle (marked by the red dashed ellipse) in 3D space and on a spherical map. If we zoom in on the surface points of the vehicle, it can be observed that three points are spatially adjacent both in 3D space and on the spherical map. Thus, the spatial relationship between different points could be directly inferred from the spherical map given the distance information of each point and corresponding index on the spherical map.

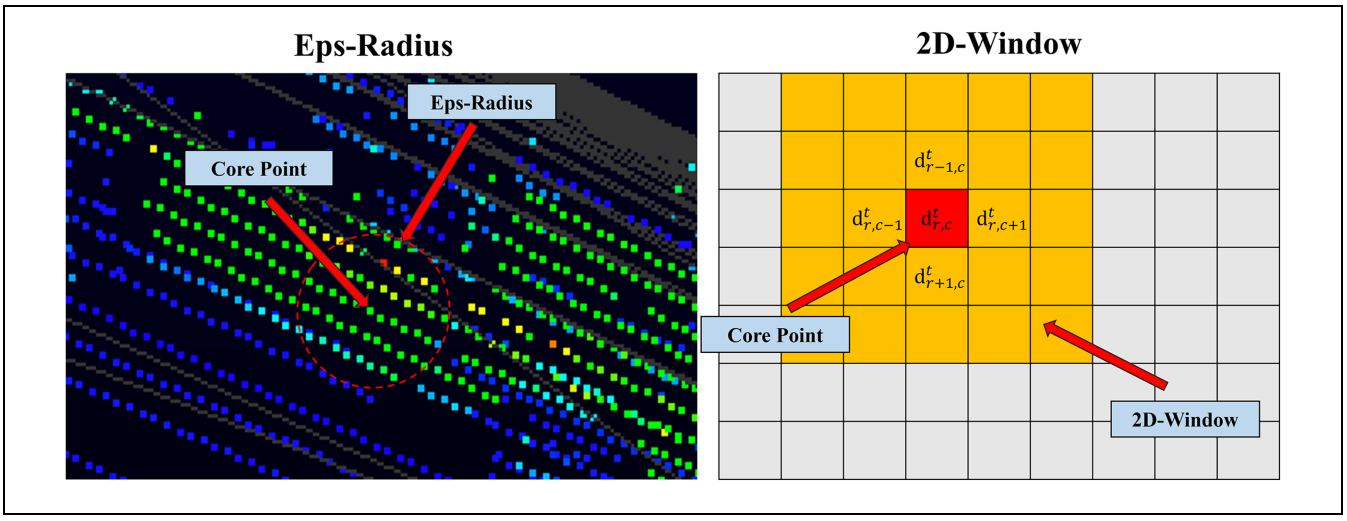

Figure 5. Comparison between eps-radius and 2D-window.
Note: DBSCAN = density-based spatial clustering of applications with noise; FSPC = fast spherical projection clustering.

For illustration purposes, a comparison of the neighborhood searching approaches of DBSCAN and FSPC is shown in Figure 5. The figure on the left is the neighborhood searching of DBSCAN which is based on the spatial eps-radius in 3D space. The figure on the right, on the other hand, is the FSPC neighborhood searching, which is based on the 2D-window searching on a spherical map. For the spatial eps-radius searching, the spatial relationship between each pair of points is first inferred from the input data by calculating the Euclidean distance or establishing the tree-based data structure such as KD-tree, which requires $O(N)$ and $O(\log(N))$ time complexity, respectively.

As shown on the right side of Figure 5, the orange area refers to the 2D-window with 5×5 size (It should be noted that the size of the 2D-window could be determined based on the demand in practice; 3×11, for example, is also viable). To query the neighborhoods using the 2D-window, two steps are required: (1) the row r and column c indices for the cells within the 2D-window are first calculated based on the central point (noted by red), and the neighbor cells within the 2D-window $\{d_{r,c} \,|r,c \in \text{window}\}$ are directly read from the spherical map according to the indices; (2) those readings whose distance $d_{r,c}$ satisfies the criterion $\left|d_{\text{center}} - d_{\text{reading}}\right| < \varepsilon$ are selected as neighborhoods.

In this criterion, the ε is another parameter for the FSPC used to exclude the window readings whose distances deviate too far from the center point, which is necessary because even though two points are close to each other on the spherical map, they may still be far apart in the 3D space. Because of the neighborhood query using the 2D-window only focusing on the cells within the window, the time complexity is independent from the whole problem of scale N, so the time complexity for 2D-window searching is $O(1)$ and the whole complexity for FSPC is $O(N)$.

The whole procedure for FSPC is given in Figure 6.

Another advantage for FSPC is that it could tackle the variation of point density problem, which would improve both accuracy and range of detection. Figure 7 shows the illustration for the point density variation problem. It can be seen that the density of the object at position A (near side) is much greater than that of the one at position B (far side), which is caused by the radiation property of LiDAR beams. For those common Euclidean-distance-based methods such as DBSCAN, OPTICS (Ordering Points to Identify the Clustering Structure), or mean-shift, the varying point density (or distance between each pair of adjacent points) may cause massive numbers of false-negative cases (target objects are unable to be detected). Taking the DBSCAN as an example, because of the difference of D_{far} and D_{near} shown in Figure 7, it is difficult for the DBSCAN to correctly cluster both far and near objects using a fixed eps-radius threshold. If the eps threshold is set large enough to cover the D_{far}, different entities at the near side may be clustered as one cluster. On the other hand, if the eps-radius is set as a small value to correctly cluster near objects as independent entities, those objects on the far side may be unable to be detected because of the sparseness.

The FSPC could avoid this problem for two reasons: (1) for those objects at the far side, even though the distance of two adjacent points is large, those two points are still near to each other on the spherical map because of the rotational order of the LiDAR sensor; and (2) the parameters in FSPC are more easily tuned to detect objects at both near and far sides correctly. The proof is provided as follows.

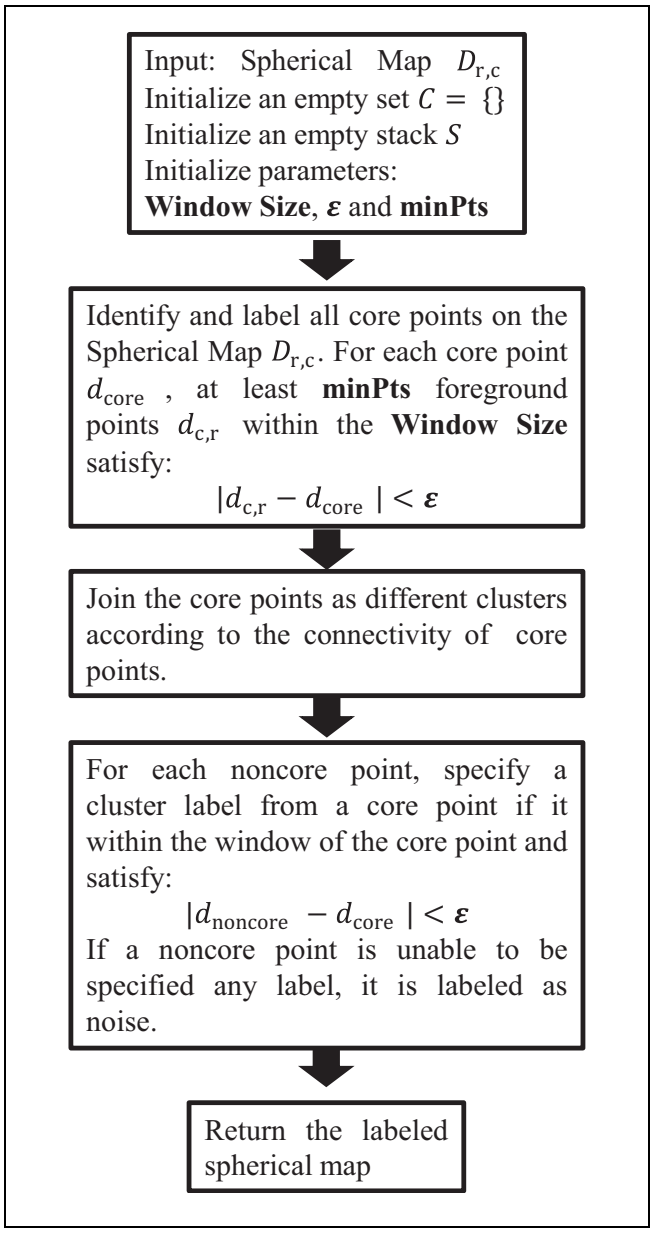

Figure 6. Flow chart of fast spherical projection clustering (FSPC).
Note: DBSCAN = density-based spatial clustering of applications with noise; FSPC = fast spherical projection clustering.

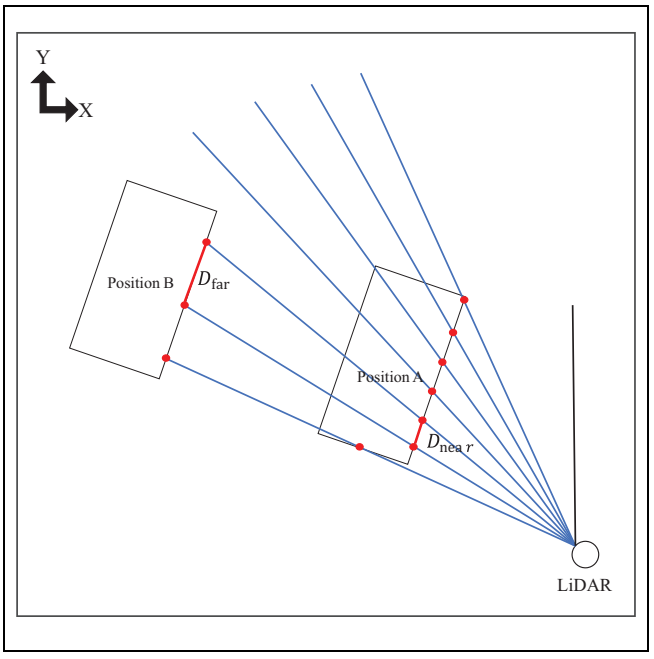

Figure 7. Illustration for point density variation (top view).
Note: DBSCAN = density-based spatial clustering of applications with noise; FSPC = fast spherical projection clustering.

Figure 8 presents an example under a generalized 2D case. The points A and B denote two adjacent points on the object surface, where the A is relatively nearer to the sensor than B because of the heading angle of the object surface, and the A' is the projection on the trace from the LiDAR sensor to point B. The d_{near} and d_{far} refer to the distance from the LiDAR sensor to points A and B, respectively. The D_{DBSCAN} and D_{FSPC} are the distance measurements for DBSCAN (Euclidean distance) and FSPC algorithm correspondingly. According to the geometric relationship, the D_{DBSCAN} and D_{FSPC} may be described by the following equations:

$$D_{DBSCAN} = d_{far}^2 + d_{near}^2 - 2d_{near}d_{far}\cos(\Delta\alpha) \quad (1)$$

$$D_{FSPC} = d_{far} - d_{near} \quad (2)$$

In this case, the $\Delta\alpha$ is a constant value representing the delta azimuth angle between two horizontal beams, which is related to the spinning frequency of the LiDAR sensor; in other words, different sensor settings would result in different values. In this study, the tested LiDAR sensor performs at a rotation frequency of 10 Hz and the $\Delta\alpha$ equals 0.2°; therefore $\cos(\Delta\alpha)$ approximates to 1 and the term $2d_{near}d_{far}\cos(\alpha)$ is approximately equal to $2d_{near}d_{far}$. Therefore the D_{DBSCAN} could be rewritten as $(d_{far} - d_{near})^2$. Assuming that $d_{far} = \gamma d_{near}$, the D_{FSPC} and D_{DBSCAN} could be recognized as

$$D_{DBSCAN} = (\gamma - 1)^2 d_{near}^2 \quad (3)$$

$$D_{FSPC} = (\gamma - 1)d_{near} \quad (4)$$

where the γ is an amplification factor deciding the heading degree of the object surface and which is greater than 1. It could be inferred that the D_{FSPC} is less sensitive than D_{DBSCAN} since the D_{FSPC} is increased linearly while the D_{DBSCAN} is increased exponentially with the distance. Thus, it is easier for FSPC to identify an appropriate threshold to separate entities at both the far and near sides since the distance measurement for FSPC only

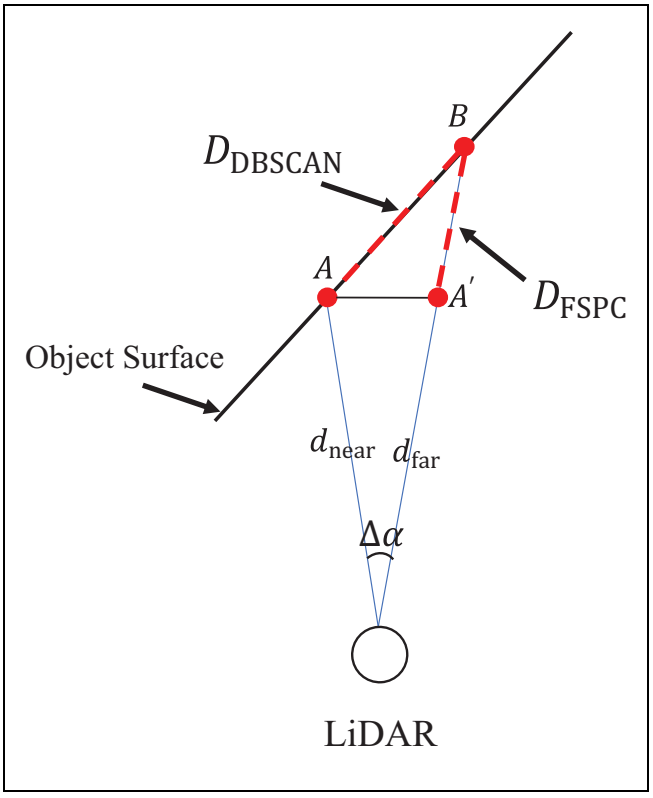

Figure 8. Illustration for point density variation.
Note: DBSCAN = density-based spatial clustering of applications with noise; FSPC = fast spherical projection clustering.

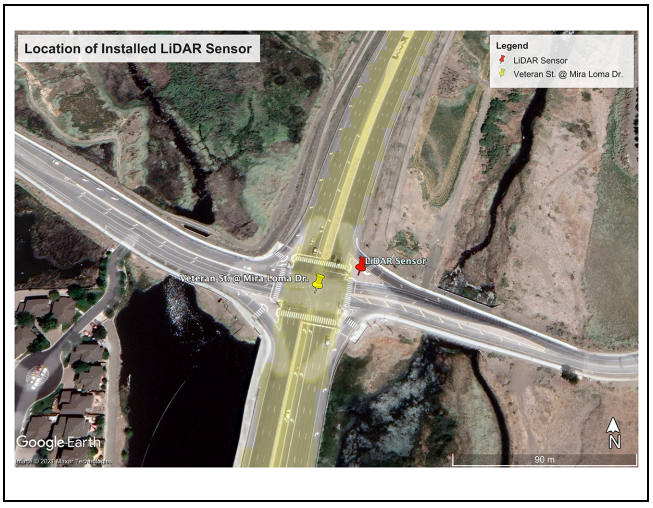

Figure 9. Data collection location.
Note: St. = Street; Dr. = Drive.

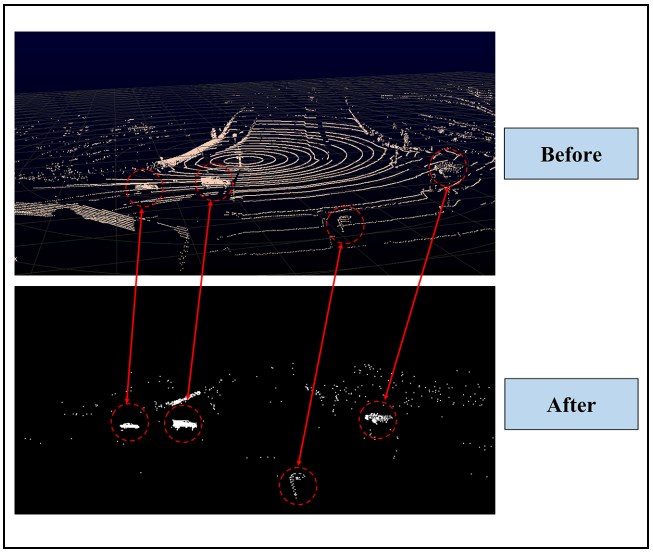

Figure 10. Comparison between before and after of background filtering.
Note: IoU = intersection-over-union.

slightly increases. For example, the Euclidean distance between adjacent points would increase 10,000 times from 1 m to 100 m, while the distance measurement for FSPC would only increase 100 times.

The output of FSPC is a labeling map with identical size to that of the spherical map, where each cell stores the point-wise instance label. In association with the spherical map, the instance-level 3D point cloud could be obtained.

Results

In this section, the FSPC is tested on a data set collected at the intersection of Veteran Street and Mira Loma Drive in Reno, Nevada (as shown in Figure 9) by CATER (Center for Advanced Transportation Education and Research), University of Nevada, Reno. The data set is collected by a 32-laser Velodyne LiDAR sensor set on a traffic signal rack (over 5 m in height). The data set includes 18,000 frames of data (half an hour). The raw data is first processed by the background filtering method provided by Wu et al. (*3*). A visualization of the post-filtered point cloud is shown in Figure 10. It can be observed that the post-filtered data still contains scattered noise, which results from the random nature of the LiDAR sensor. The noise will be identified and excluded by the clustering algorithm. Next, 2000 frames of the collected data are manually labeled with the point-wise label (target object or background).

On average, 1699 foreground points are contained in each frame, and the foreground points would surge to over 10,000 at congested traffic conditions. Approximately 40% of points are scattered background points, which are usually dynamic background points such as trees and bushes that failed to be filtered by background filtering algorithms; these need to be further excluded by clustering. We first tune the proposed FSPC on the labeled data set to obtain the optimal parameters.

Table 1. Parameter Searching Range

Parameter	Algorithm	Searching range
Eps	FSPC	[0.5,0.8,1.0,1.2,1.5,1.8]
	DBSCAN	[0.5,0.8,1.0,1.2,1.5,1.8,2.0,2.2,2.4,2.6,2.8,3.0]
minPts	FSPC	[10,15,20,25,30]
	DBSCAN	
Height of window	FSPC	[2,3,4,5,6,7,8]
Width of window		[7,9,11,13,15,17,19,23,27]

Note: FSPC = fast spherical projection clustering; DBSCAN = density-based spatial clustering of applications with noise; minPts = minimum points for a cluster); eps = the radius to search neighborhoods.

Second, the scores including accuracy and time consumption of FSPC are obtained through the labeled data set, which is also compared with existing typical clustering algorithms DBSCAN and a CCL-based algorithm. Last, the FSPC with the optimal parameters is tested on the whole data set to validate the detection range and time complexity. The whole experiment is performed on a desktop PC with an Intel Core i5-7500 3.40GHz processor and 16GB RAM. All implementations including algorithm coding and experiment are carried out under the Python 3.8 environment.

As mentioned in the methodology, three parameters: ε (eps), size of the 2D window, and minPts are involved in the proposed FSPC. For evaluation purposes, the best combination of different parameters is first tuned to obtain optimal performance. To further justify the performance of the proposed FSPC, two widely applied unsupervised clustering algorithms are included for comparison: DBSCAN and CCL-based (*15*). All three clustering algorithms satisfy the "cluster number free" requirement; thus they could be fairly compared.

Since we only focus on the detection for target objects (all road users), the total classes in the clustered results consist of dynamic background (noise) and target points (foreground). In this work, metrics for image segmentation are selected for the performance evaluation, because the output (labeling map) has the same property as the labeling mask in the image segmentation tasks. The selected metrics include precision, recall, and IoU (intersection-over-union). The point-wise ground truth labels are compared with the output of FSPC and the metrics are calculated. The calculation could be defined as

$$\begin{aligned} \text{precision}_c &= \frac{|O_c \cap G_c|}{|O_c|}, \\ \text{recall}_c = \frac{|O_c \cap G_c|}{|G_c|}, &\text{IoU}_c = \frac{|O_c \cap G_c|}{|O_c \cup G_c|} \end{aligned} \tag{5}$$

where O_c and G_c respectively refer to the output and ground truth point sets belonging to class c. The symbol $|\cdot|$ denotes the cardinality (number of points) of the point set.

The parameter searching range determined empirically is presented in Table 1; by this, all possible combinations of the different parameters are tested.

The tuning starts from the parameter minPts. Figure 11 presents the average score of metrics given different minPts settings. For dynamic background, three accuracy metrics are maintained at high scores over 99.7% regardless of different minPts values. While, for the target points, the metrics would change differently according to different minPts settings. The precision value reached the highest score at 20 minPts, the IoU metric reached the summit at 15 minPts, and the recall kept dropping in the whole range. Given the application of roadside LiDAR, we hope to detect as many objects as possible and ensure detection precision. Therefore, the optimal value of minPts should be 15.

Given 15 as the minPts setting, the parameter setting of eps is tested similarly. Figure 12 presents the average metrics given different eps. Similar to the previous test, the metrics of dynamic background are also maintained at a high level. For the target points, the precisions decline, with a tiny trend from 95.1% to 94.3%. The recall and IoU uniformly improve in the whole eps range, reaching 98.0% and 92.5%, respectively. Thus, we consider the best eps as 1.8.

After that, the size of the searching window is tested, which includes two parameters: height and width. The window size will significantly affect the searching range and change computation load; therefore, we tested it in association with the time consumption. Figures 13 and 14 show the tested height and width respectively, and Figure 15 shows the variation of time consumption. Again, the impact of different parameters on the dynamic background can be ignored. As shown in Figure 13, the precision and IoU are affected significantly by the window width: a longer window width contributes to a lower metric value after the width increases to 11, with them finally decreasing to 86.8% and 85.8%, respectively, while the recall keeps increasing through the whole range. As shown in Figure 14, the height parameter presents a similar trend, the recall value for target points converging at 5. Additionally, as shown in Figure 15, the time consumption increases linearly as the width increases.

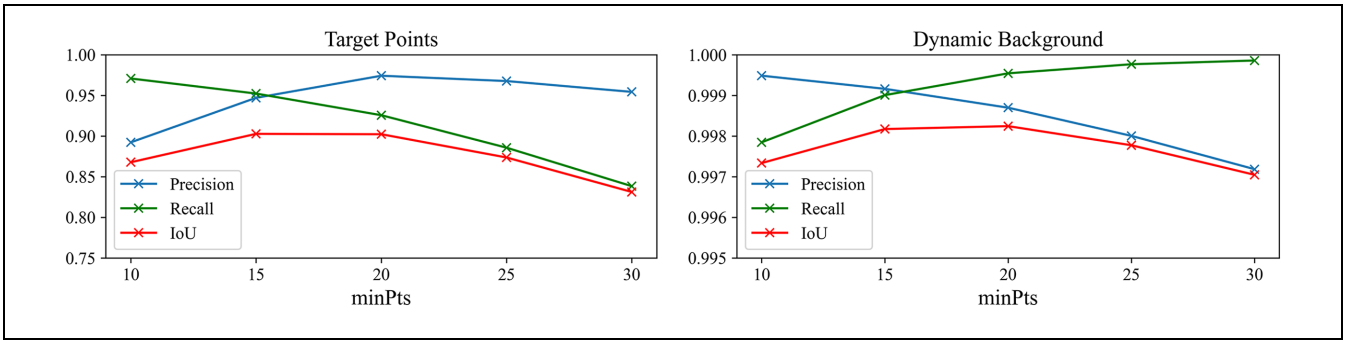

Figure 11. Three metrics given different minPts.
Note: IoU = intersection-over-union.

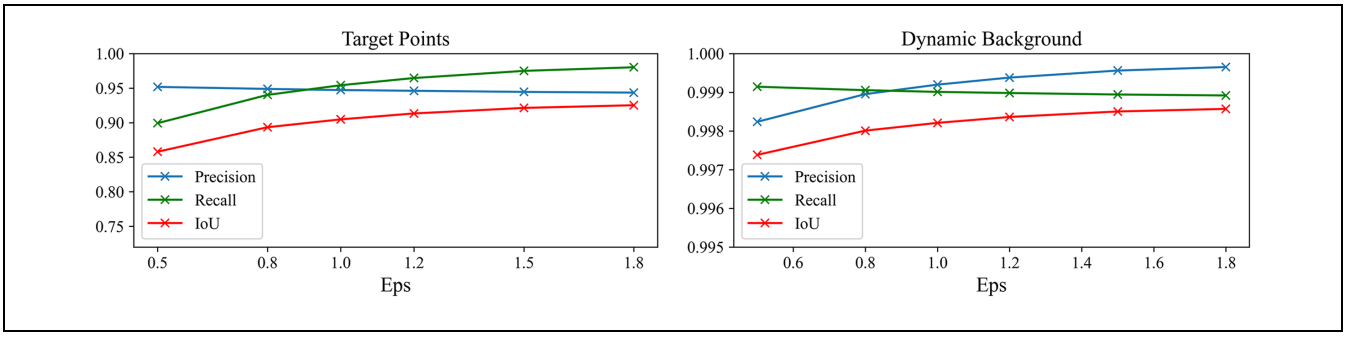

Figure 12. Three metrics given different eps.
Note: IoU = intersection-over-union.

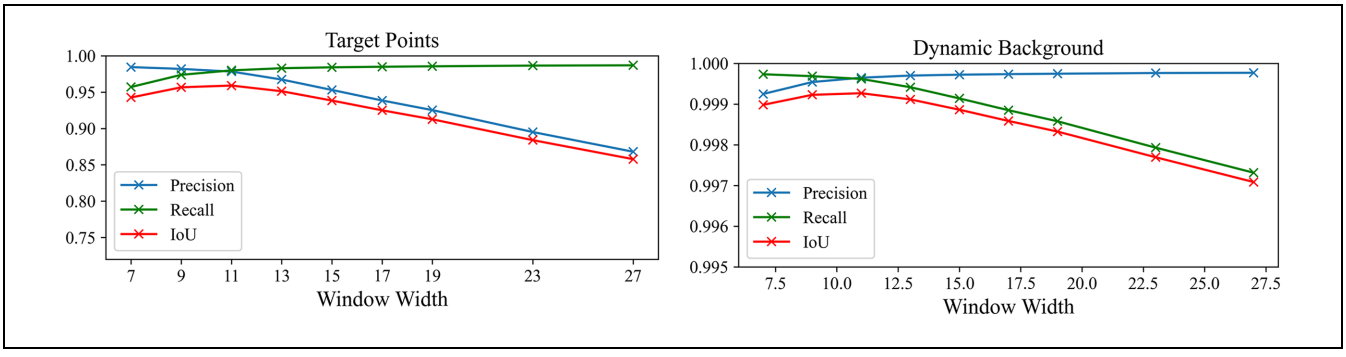

Figure 13. Three metrics given different window width.
Note: IoU = intersection-over-union.

Taking all things together, a window width of 11 is selected and, based on a similar analysis, a window height of 5 is chosen.

Based on the tuning test on different parameters, the optimal parameter for FSPC is summarized in Table 2. The same tuning test is implemented on the DBSCAN, and the result is also given in Table 2. It can be seen that the "eps" parameter for FSPC is higher than for DBSCAN, the reason for this result could be the difference between 3D and 2D distance measurement, which is proved in the previous section.

The overall summary of metrics based on the best parameters setting is given in Table 3. In addition, an instance-level error between the number of detected objects and ground truth is introduced to adequately reveal the performance. The instance-level error is calculated by counting the instances that have instance-IoU over 50% between output and ground truth. From Table 3, it is observed that the dynamic background points could be easily identified by three methods. All three methods could obtain high accuracy metrics (precision, recall, and IoU) over 98%. However, the scores for

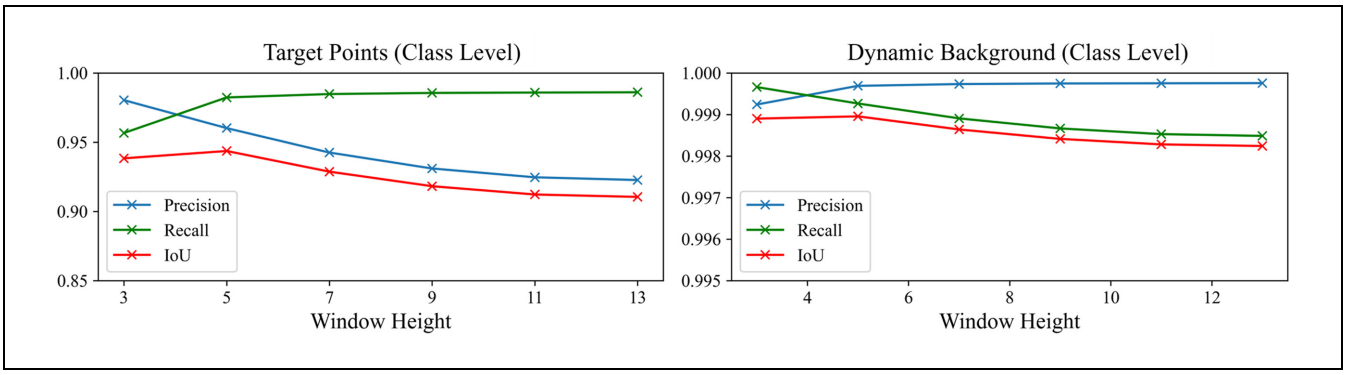

Figure 14. Three metrics given different window height.
Note: IoU = intersection-over-union.

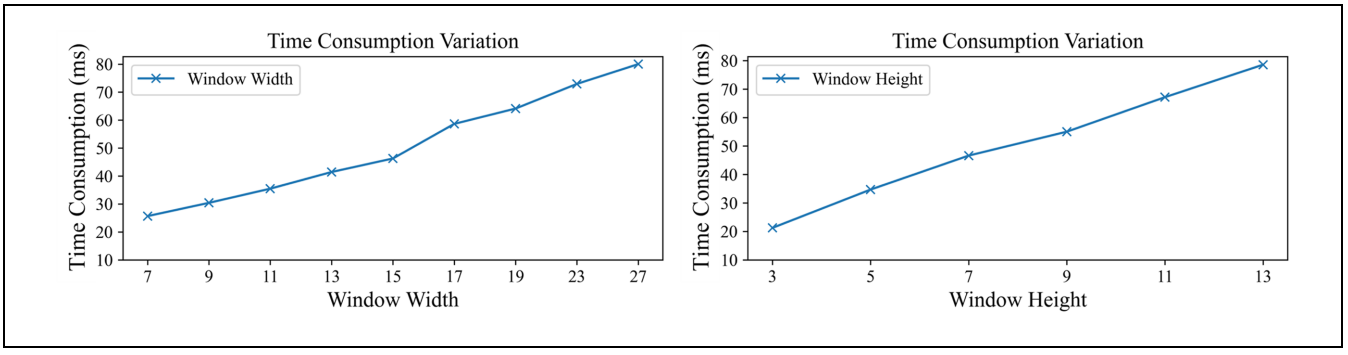

Figure 15. Variation of time consumption given different window size.
Note: IoU = intersection-over-union.

Table 2. Optimal Parameters for Clustering Algorithm

Method	minPts	Eps	Window size
FSPC	15	1.8	(5 × 11)
DBSCAN	15	1.2	na

Note: FSPC = fast spherical projection clustering; DBSCAN = density-based spatial clustering of applications with noise; minPts = minimum points for a cluster); eps = the radius to search neighborhoods. the Window size is not applicable for DBSCAN.

target point identification are dramatically different across the three methods. For the target point identification, it could be seen that the highest recall score is achieved by the CCL-based method with 99.7% and followed by the proposed FSPC and DBSCAN with 98.3% and 96.7% correspondingly. However, the precision score obtained by the CCL-based method is the lowest among the three methods with only 66.8%. Additionally, the IoU score for the CCL-based method is also the lowest with 66.7%. Two reasons contributed to the low

Table 3. Performance Evaluation Result

Method	Class	Precision (%)	Recall (%)	IoU (%)	μ_{cost} (ms)	σ_{cost} (ms)	Matched instances (IoU > 0.5)
FSPC	Target point	98.2	98.3	96.5	24.4	2.36	1483/1511
	Dynamic background	99.9	99.9	99.9			
DBSCAN	Target point	82.3	96.7	80.0	117.1	6.59	1414/1511
	Dynamic background	99.9	99.6	99.5			
CCL	Target point	66.8	99.7	66.7	7.8	2.86	787/1511
	Dynamic background	99.9	99.6	98.7			

Note: FSPC = fast spherical projection clustering; DBSCAN = density-based spatial clustering of applications with noise; CCL = connected-component labeling; IoU = intersection-over-union.

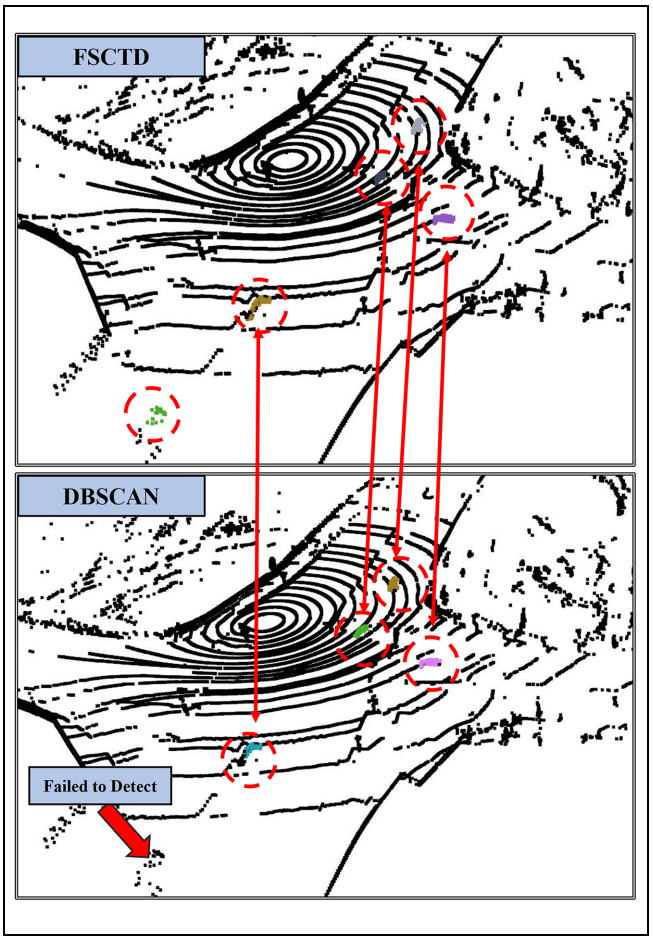

Figure 16. Results for different clustering algorithm.
Note: FSPC = Fast-spherical-projection-based point cloud clustering; DBSCAN = density-based spatial clustering of applications with noise.

accuracy of the CCL-based method: 1) Due to the occlusions or packet loss are frequently occurring in the LiDAR applications, the CCL-based method is unable to handle the point disconnections. 2) the CCL-based method doesn't consider the noise handling in the clustering scheme For the proposed FSPC and DBSCAN, they both applied a density-based strategy to identify the object, thus they could alleviate the impact from the point disconnection and noise points, and further improve the detection accuracy compared to the CCL-based method. Between the FSPC and DBSCAN, the precision score of the DBSCAN (82.3%) is lower than the proposed FSPC (98.2%), and its matched instance (1414 over 1511) is also less than the proposed FSPC (1483 over 1511), which suggests that the DBSCAN would generate more false-positive cases compared to FSPC. The higher precision and matched instances for the proposed method are attributed to the better handling of density variation issue by the 2D-window neighborhood searching strategy in the FSPC. Most LiDAR data processing application usually requires a processing speed greater than 100 ms/frame to satisfy the real-time processing requirement. In the three methods, the FSPC and CCL-based methods could satisfy the real-time level with 24.4 ms/frame and 7.8 ms/frame respectively. The CCL-based method is faster than the FSPC because it requires less computation consumption. However, the FSPC has better performance on the accuracy metrics, which is considered to be prioritized in the roadside LiDAR applications. Additionally, the processing speed of FSPC is still under the real-time level with only 24% of the real-time requirement and its stability is greatest with the lowest standard deviation of 2.36 ms. Comprehensively considering the score of different metrics and time consumption, the proposed method could outperform other methods in the LiDAR data processing tasks.

The variant density problem is settled by the 2D-window searching strategy, which was introduced in the previous section. Figure 16 shows a 3D visualization of this improvement. It can be observed that FSPC detected five objects while DBSCAN only detected four objects. The missing object for DBSCAN locates at the lower-left corner which has a sparser point cloud than others. This is because the eps-radius of DBSCAN could detect those objects that are near to the LiDAR while it failed to detect those far from the LiDAR. The overview of all detected objects of the three algorithms is presented in Figure 17, each green dot representing the center of a detected cluster. It can be observed that the three algorithms are able to profile the outline of the testing intersection and the width of lanes. Although the CCL-based algorithm could maximally detect all objects in the intersection, there is also a great deal of noise in the scene. Both FSPC and DBSCAN could correctly detect the objects in each lane compared with the CCL-based algorithm, while it can be observed that the DBSCAN failed to detect objects circled by the red dashed line, which are the objects located over 100 m away. For the FSPC, the remotest detected object is located over 200 m away.

For roadside LiDAR, the robustness to variant computation loads is important. This is because the foreground points will periodically surge because of traffic features such as regular traffic peaks. To address this concern, we further tested the impact of the different numbers of foreground points on the algorithm efficiency. Figure 18 shows the foreground points' accumulative proportion across 18,000 frames in the collected intersection. From the figure, it can be reported that over 90% of frames contain fewer than 10,000 foreground points and approximately 10% of frames have more than 10,000 foreground points, up to 23,000 points. Figure 19 reveals the correlation between the foreground points and the time efficiency of clustering for different algorithms. Apparently, the time consumption of DBSCAN

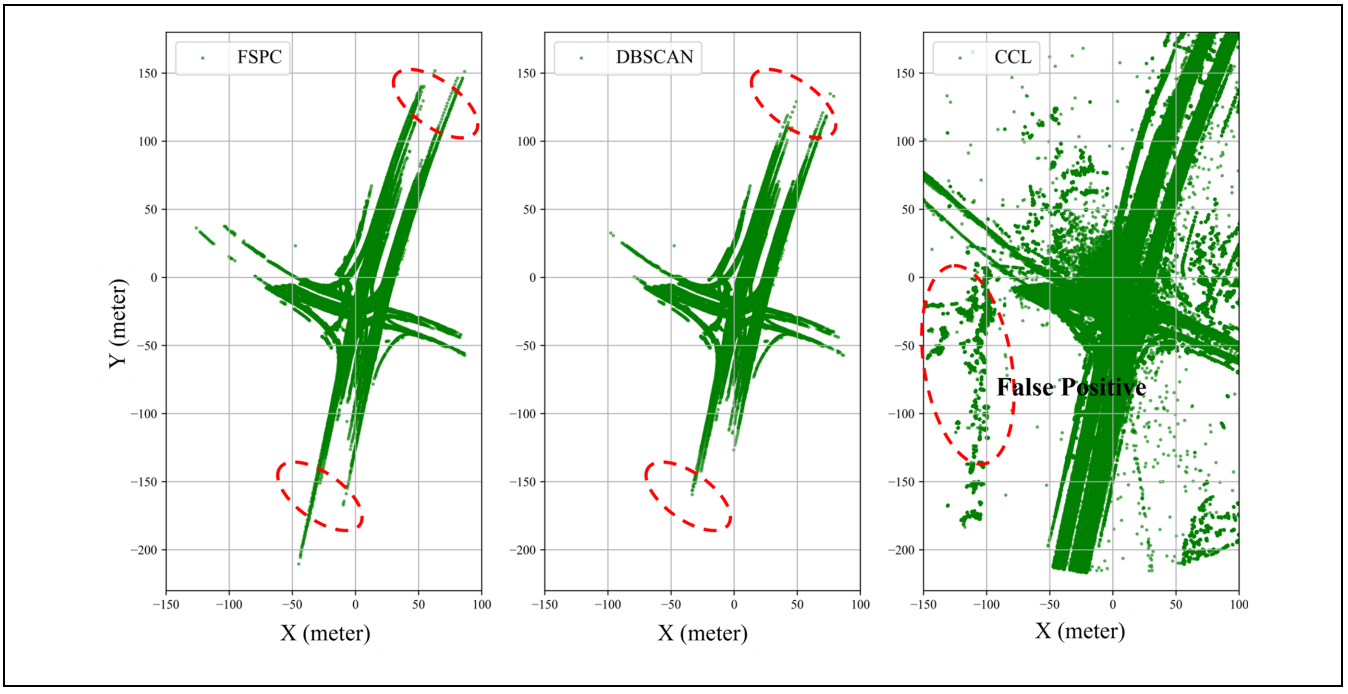

Figure 17. Comparison for different detection range.
Note: FSPC = fast spherical projection clustering; DBSCAN = density-based spatial clustering of applications with noise; CCL = connected-component labeling.

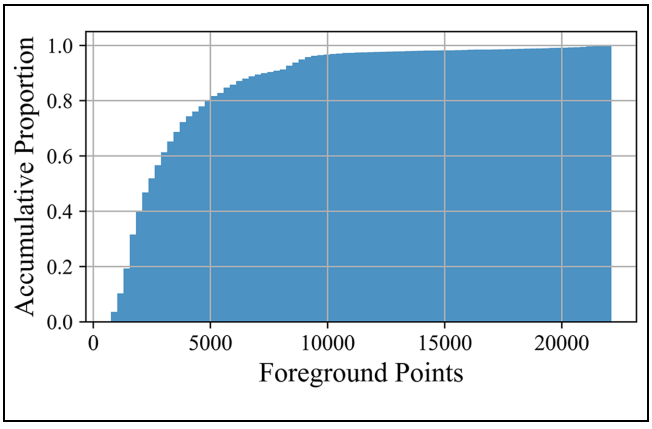

Figure 18. Accumulative proportion of foreground points.

increases significantly with the increment of foreground points, whereas the FSPC and CCL-based algorithms only increase to a tiny extent and the increasing trend for the CCL-based algorithm is lower than that for the FSPC. This is because the procedure of the CCL-based algorithm is simpler than that for FSPC. The maximum time consumption for DBSCAN would reach 2500 ms in the most complex scene, which would be devastating to the real-time application. As for the proposed FSPC, the maximum time consumption would not exceed 85 ms under the same foreground points, as shown in Figure 20. For the most common scene—that the foreground points are fewer than 10,000—the DBSCAN would consume up to 500 ms, while the FSPC is maintained below 40 ms in most situations, which fulfills the requirement for real-time data processing under 100 ms per frame.

Note that the DBSCAN would outperform the FSPC when the foreground points are fewer than 2000. This is because the FSPC requires more computational load during the preprocessing, which is related to the coding implementation; as a consequence, the time for preprocessing exceeds the time for the clustering process.

In summary, the FSPC still reaches linear time consumption and could be robust in over 90% of situations.

Conclusions

In this work, we proposed a fast clustering algorithm FSPC to efficiently and accurately cluster variant-density LiDAR point clouds. The proposed algorithm utilizes the advantage of spherical map and a novel 2D-window searching to accelerate the clustering process, and further improves the clustering accuracy compared with existing algorithms. The testing result showed that (1) the proposed FSPC could achieve within 40 ms per frame in the most common scenes (foreground points within 5000); and (2) over 96% of detected objects have over 50% IoU with instance-level ground truths, and over 98% in both precision and recall metrics—compared with traditional

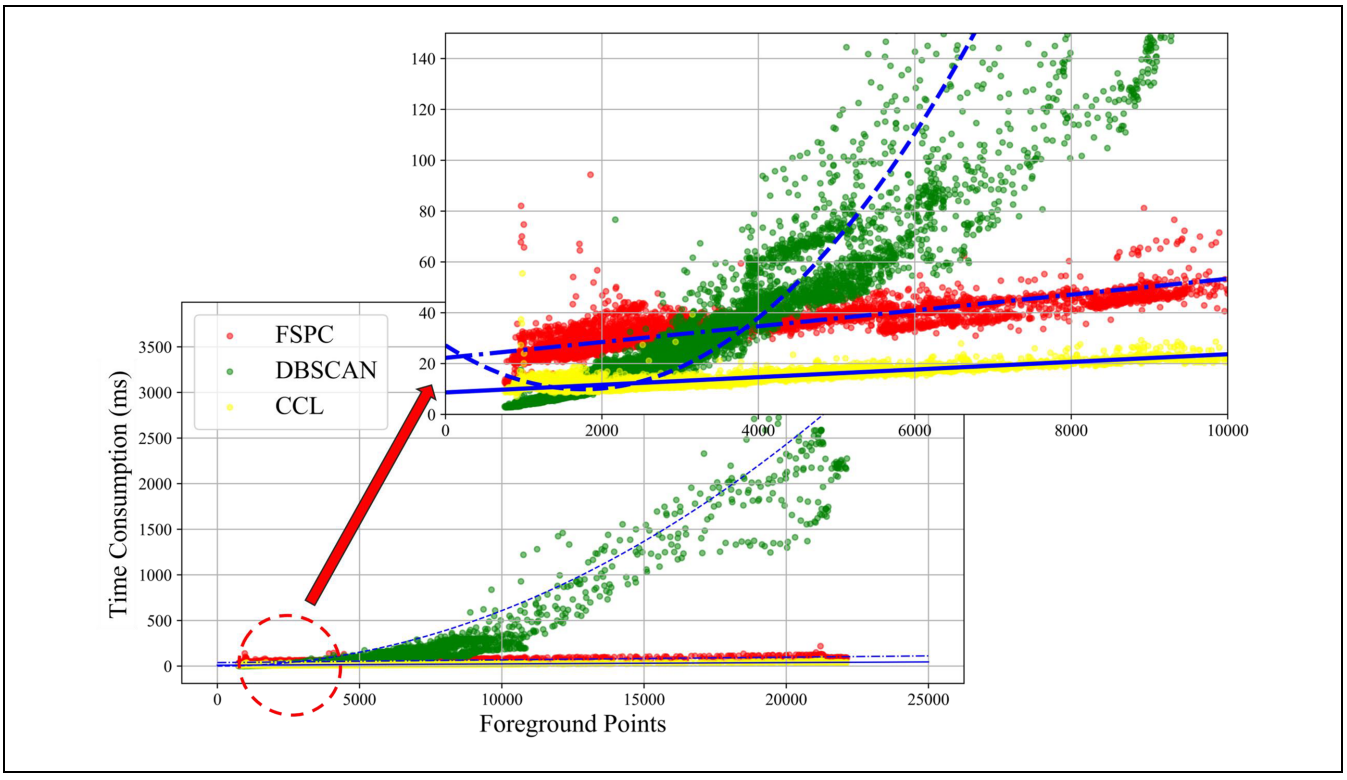

Figure 19. Increment trend of time consumption for clustering algorithms.
Note: FSPC = fast spherical projection clustering; DBSCAN = density-based spatial clustering of applications with noise; CCL = connected-component labeling.

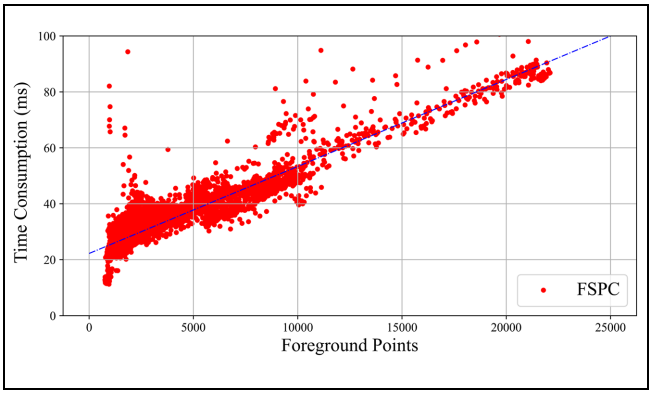

Figure 20. Increment trend of time consumption for FSPC.
Note: FSPC = fast spherical projection clustering.

DBSCAN and CCL-based algorithms, the FSPC outperformed them, with all accuracy metrics over 96%. As for the time consumption, the FSPC is 4.8 times faster than the DBSCAN and could satisfy the real-time requirement. The process time would be maintained within 40 ms in over 90% of frames in the testing site. Even though the FSPC is slightly slower than the CCL-based method, it is overall greater than both DBSCAN and the CCL-based algorithm in the accuracy metrics. Additionally, we also identified a wider detection range, up to 200 m, which is 33% further than traditional DBSCAN. The proposed FSPC is expected to be applied in the field to improve the roadside LiDAR system. Future work will include background filtering based on the spherical map and a multi-object tracking algorithm to be added before and after the FSPC clustering algorithm, constituting the whole roadside LiDAR processing pipeline. Moreover, the generality of FSPC enables the proposed algorithm to be applied in other areas such as autonomous driving or remote sensing. Finally, we also believe the efficiency of FSPC could be further improved by optimizing the code implementation.

Acknowledgments

The authors gratefully acknowledge the financial support and are thankful to the engineers from the Nevada Department of Transportation (NDOT) and the City of Reno for helping develop the testbeds.

Author Contributions

The authors confirm contribution to the paper as follows: study conception and design: Z. Chen, H. Xu; data collection: H. Xu; analysis and interpretation of results: J. Zhao, Z. Chen; draft manuscript preparation: Z. Chen, J. Zhao, H. Xu, H. Liu. All authors reviewed the results and approved the final version of the manuscript.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Nevada Department of Transportation (NDOT) under Grant No. P744-18-803.

ORCID iDs

Zhihui Chen https://orcid.org/0000-0001-9893-3009
Hao Xu https://orcid.org/0000-0003-1314-4540
Junxuan Zhao https://orcid.org/0000-0001-9927-7023
Hongchao Liu https://orcid.org/0000-0001-7092-9606

Data Accessibility Statement

Some or all data, models, or code that support the findings of this study are available from the corresponding author on reasonable request.

References

1. Lv, B., H. Xu, J. Wu, Y. Tian, and C. Yuan. Raster-Based Background Filtering for Roadside LiDAR Data. *IEEE Access*, Vol. 7, 2019, pp. 76779–76788.
2. Ilgin Guler, S., M. Menendez, and L. Meier. Using Connected Vehicle Technology to Improve the Efficiency of Intersections. *Transportation Research Part C: Emerging Technologies*, Vol. 46, 2014, pp. 121–131. http://doi.org/10.1016/j.trc.2014.05.008.
3. Wu, J., H. Xu, and J. Zheng. Automatic Background Filtering and Lane Identification with Roadside LiDAR Data. *Proc., IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, IEEE, New York, March 15, 2018, pp. 1–6.
4. Wu, J., h. xu, R. Sun, and P. Zhuang. Road Boundary-Enhanced Automatic Background Filtering for Roadside LiDAR Sensors. *IEEE Intelligent Transportation Systems Magazine*, 2021, pp. 2–14. https://doi.org/10.1109/MITS.2021.3049358.
5. Zhao, J., H. Xu, H. Liu, J. Wu, Y. Zheng, and D. Wu. Detection and Tracking of Pedestrians and Vehicles Using Roadside LiDAR Sensors. *Transportation Research Part C: Emerging Technologies*, Vol. 100, 2019, pp. 68–87. https://doi.org/10.1016/j.trc.2019.01.007.
6. Chen, J., H. Xu, J. Wu, R. Yue, C. Yuan, and L. Wang. Deer Crossing Road Detection with Roadside LiDAR Sensor. *IEEE Access*, Vol. 7, 2019, pp. 65944–65954.
7. Wang, C., M. Ji, J. Wang, W. Wen, T. Li, and Y. Sun. An Improved DBSCAN Method for LiDAR Data Segmentation with Automatic Eps Estimation. *Sensors (Basel, Switzerland)*, Vol. 19, No. 1, 2019, p. 172.
8. Zhao, J., Y. Dong, S. Ma, H. Liu, S. Wei, R. Zhang, and X. Chen. An Automatic Density Clustering Segmentation Method for Laser Scanning Point Cloud Data of Buildings. *Mathematical Problems in Engineering*, Vol. 2019, 2019. https://doi.org/10.1155/2019/3026758.
9. Velodyne LiDAR, Inc. *VLS-128 User Manual 63-9483*. Velodyne LiDAR, Inc., San Jose, CA.
10. Himmelsbach, M., T. Luettel, and H. J. Wuensche. Real-Time Object Classification in 3D Point Clouds Using Point Feature Histograms. *Proc., IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, IEEE, New York, 2009, pp. 994–1000.
11. Suzuki, K., I. Horiba, and N. Sugie. Linear-Time Connected-Component Labeling Based on Sequential Local Operations. *Computer Vision and Image Understanding*, Vol. 89, No. 1, 2003, pp. 1–23.
12. John, V., Q. Long, Z. Liu, and S. Mita. Automatic Calibration and Registration of Lidar and Stereo Camera without Calibration Objects. *Proc., IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Yokohama, Japan, IEEE, New York, 2016, pp. 231–237.
13. Ravankar, A., Y. Kobayashi, A. Ravankar, and T. Emaru. A Connected Component Labeling Algorithm for Sparse Lidar Data Segmentation. *Proc., 6th International Conference on Automation, Robotics and Applications (ICARA)*, Queenstown, New Zealand, IEEE, New York, 2015, pp. 437–442.
14. Börcs, A., B. Nagy, and C. Benedek. Fast 3-D Urban Object Detection on Streaming Point Clouds. In Computer Vision—ECCV 2014 Workshops, Vol. 8926. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Agapito, L., M. Bronstein, and C. Rother, eds), Springer, Cham, Switzerland, 2015, pp. 628–639.
15. Zermas, D., I. Izzat, and N. Papanikolopoulos. Fast Segmentation of 3D Point Clouds: A Paradigm on LiDAR Data for Autonomous Vehicle Applications. *Proc., IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, IEEE, New York, 2017, pp. 5067–5073.
16. Bogoslavskyi, I., and C. Stachniss. Fast Range Image-Based Segmentation of Sparse 3D Laser Scans for Online Operation. *Proc., IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea (South), IEEE, New York, 2016, pp. 163–169.
17. He, L., X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao. The Connected-Component Labeling Problem: A Review of State-of-the-Art Algorithms. *Pattern Recognition*, Vol. 70, 2017, pp. 25–43.
18. He, L., Y. Chao, and K. Suzuki. A Run-Based Two-Scan Labeling Algorithm. *IEEE Transactions on Image Processing*, Vol. 17, No. 5, 2008, pp. 749–756.
19. Hasecke, F., L. Hahn, and A. Kummert. FLIC : Fast Lidar Image Clustering. *arXiv Preprint arXiv:2003.00575*, 2020.