

Article

Grid-Based DBSCAN Clustering Accelerator for LiDAR's Point Cloud

Sangho Lee , Seongmo An , Jinyeol Kim , Hun Namkung , Joungmin Park , Raehyeong Kim 
and Seung Eun Lee * 

Department of Electronic Engineering, Seoul National University of Science and Technology,
Seoul 01811, Republic of Korea; leesangho@seoultech.ac.kr (S.L.); ahnseongmo@seoultech.ac.kr (S.A.);
kimjinyeol@seoultech.ac.kr (J.K.); namkunghun@seoultech.ac.kr (H.N.); parkjoungmin@seoultech.ac.kr (J.P.);
kimraehyeong@seoultech.ac.kr (R.K.)

* Correspondence: seung.lee@seoultech.ac.kr; Tel.: +82-2-970-9021

Abstract: Autonomous robots operate on batteries, rendering power efficiency essential. The substantial computational demands of object detection present a significant burden to the low-power cores employed in these robots. Therefore, we propose a grid-based density-based spatial clustering of applications with a noise (DBSCAN) clustering accelerator for light detection and ranging (LiDAR)'s point cloud to accelerate computational speed and alleviate the operational burden on low-power cores. The proposed method for DBSCAN clustering leverages the characteristics of LiDAR. LiDAR has fixed positions where light is emitted, and the number of points measured per frame is also fixed. These characteristics make it possible to impose grid-based DBSCAN on clustering a LiDAR's point cloud, mapping the positions and indices where light is emitted to a 2D grid. The designed accelerator with the proposed method lowers the time complexity from $O(n^2)$ to $O(n)$. The designed accelerator was implemented on a field programmable gate array (FPGA) and verified by comparing clustering results, speeds, and power consumption across various devices. The implemented accelerator speeded up clustering speeds by 9.54 and 51.57 times compared to the i7-12700 and Raspberry Pi 4, respectively, and recorded a 99% reduction in power consumption compared to the Raspberry Pi 4. Comparisons of clustering results also confirmed that the proposed algorithm performed clustering with high visual similarity. Therefore, the proposed accelerator with a low-power core successfully accelerated speed, reduced power consumption, and effectively conducted clustering.

Keywords: light detection and ranging (LiDAR); density-based spatial clustering of applications with noise (DBSCAN); hardware accelerator; clustering



Citation: Lee, S.; An, S.; Kim, J.; Namkung, H.; Park, J.; Kim, R.; Lee, S.E. Grid-Based DBSCAN Clustering Accelerator for LiDAR's Point Cloud. *Electronics* **2024**, *13*, 3395. <https://doi.org/10.3390/electronics13173395>

Academic Editors: Mohammad Awrangzeb and Fayez Tarsha Kurdi

Received: 31 July 2024

Revised: 18 August 2024

Accepted: 23 August 2024

Published: 26 August 2024



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

1. Introduction

With advancements in object detection through various sensors, autonomous driving technology has been studied for driverless driving, anticipated to alleviate human labor in the future [1–4]. Additionally, with the development of internet of things (IoT) technology, tasks like serving and delivery are now being replaced by autonomous robots [5,6]. These robots recognize objects and avoid obstacles with the help of multiple sensors to reach their destinations. A common characteristic among these autonomous robots is their reliance on batteries. Because of the limited capacity of the batteries, power efficiency becomes crucial for robots. Additionally, applying many sensors or power-intensive cores is a significant burden for the development of autonomous robots [7,8]. Therefore, autonomous robots adopt energy-efficient cores or sensors designed for low-power devices. However, low-power cores generally offer inferior performance compared to standard cores, posing a challenge for executing algorithms that require extensive computation on autonomous robots.

In autonomous driving, sensors employed for object detection include camera sensors, also called image sensors, radio detection and ranging (RADAR) sensors, ultrasonic sensors,

and light detection and ranging (LiDAR) sensors. Among them, The camera and LiDAR are sensors that recognize objects on their own without the need for other sensors. These sensors, based on high resolution, allow for the identification of the specific size and shape of objects. Camera sensors recognize objects based on 2D planar RGB images, with artificial intelligence technologies such as regions with convolutional neural networks features (RCNN) and you only look once (YOLO), which is commonly applied in object detection [9–12]. LiDAR works by emitting light and measuring the time it takes for the reflected light to return, thus determining the location of objects. By emitting and detecting multiple light beams simultaneously, LiDAR identifies the reflected positions of these beams and represents them as points, which collectively form a point cloud. This point cloud consists of three-dimensional data with x, y, and z coordinates. RADAR sensors utilize radio waves for object recognition, offering longer detection distances and greater robustness in adverse weather conditions compared to LiDAR. However, they exhibit lower resolution and a narrower viewing range than LiDAR. Consequently, RADAR is often fused with other sensors to enhance performance in object recognition tasks [13–16]. Camera sensors struggle with object detection in dark or unlit areas, whereas LiDAR, which measures distance with light, performs better in such environments but experiences issues with light interference in bright conditions. Additionally, both sensors do not function properly when objects are visually obscured or when light is blocked or scattered due to adverse weather conditions. Consequently, cameras, LiDAR, RADAR, and other sensors adopted in autonomous driving work together to complement each other's weaknesses [16,17].

There are two methods for object detection with LiDAR: one is supervised learning such as deep learning [18] and support vector machine (SVM) [19], and the other relies on unsupervised learning such as clustering [20]. Among various methods, Clustering forms clusters or groups based on the positions of points and the distances between them, allowing for the detection of untrained obstacles.

Object detection through clustering LiDAR's point clouds has been conducted with various clustering algorithms, including K-means clustering [21], hierarchical clustering [22], and several density-based clustering algorithms [23]. K-means clustering has a relatively low time complexity of $O(n)$ but requires a predefined number of clusters, which is a hindrance in autonomous driving where the number of objects encountered is unpredictable. Hierarchical clustering allows for easy modification of the number of clusters, but it has a high time complexity of $O(n^3)$. Among density-based clustering methods, density-based spatial clustering of applications with noise (DBSCAN) is commonly applied for clustering with LiDAR's point clouds [24,25]. The DBSCAN algorithm clusters based on the density of points, treating areas with sparse points as noise, which means clustering does not include all points in clusters and vary the number of clusters formed. With these characteristics of the algorithm, DBSCAN, an unsupervised learning method, detects untrained objects in varying situations and selectively clusters only obstacles, providing significant advantages in autonomous environments.

Figure 1 illustrates stages of the DBSCAN clustering algorithm. First, a random point is selected, and the distances to nearby points are measured. Based on computed distances, it is determined whether a certain number of points (*MinPts*) exist within a specific radius (*eps*) from the core point. If the number of points within the radius exceeds *MinPts*, the core point and the points within this distance form one group. Once a group is formed, another point within the group is selected and the process repeats, measuring distances to nearby points and deciding whether to form a new group. If a new group is formed, it merges with the existing group to which the core point belongs. If not, another point within the group is chosen to search for possible group formation. This process of checking each point in the group continues until all possible group formations have been searched, concluding the merging for that group. Once one group is fully formed, the process starts again with a random point that has not been included in any group. This continues until all points have been checked for group formation, at which point the clustering process is completed.

The DBSCAN Clustering algorithm, with a time complexity of $O(n^2)$, combined with the fact that LiDAR's point cloud data contain numerous 3D points, means that object detection with LiDAR involves substantial computational effort. Consequently, such machine learning-based object detection methods are primarily conducted in environments capable of processing a large number of operations quickly. Moreover, in autonomous driving, object detection methods compute and deliver results quickly based on real-time data, making processing speed even more critical. However, for small autonomous robots like delivery robots that embed low-power cores to conserve battery efficiency, this high volume of computation not only presents a burden in terms of power, but also results in processing speeds slow enough to hinder real-time object detection. Therefore, small delivery robots require a reduction in the amount of computation involved in object detection or need separate accelerators to efficiently speed up processing in terms of power and time.

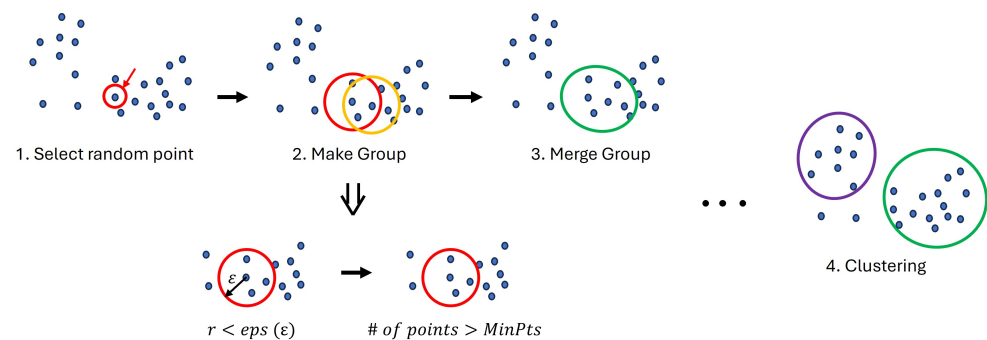


Figure 1. Stages of DBSCAN clustering algorithm.

This paper proposes a LiDAR point cloud DBSCAN clustering accelerator connected to a low-power core. The proposed DBSCAN clustering accelerator integrates the previously mentioned DBSCAN algorithm to enhance computational speed and offloads the computational workload from the core. The DBSCAN algorithm applied in the accelerator employs a method of clustering based on indices, which is optimized for solid-based LiDAR, and the computational load of the accelerator has been reduced through quantization. The DBSCAN clustering accelerator was evaluated by interfacing it with a Cortex-M0 to verify its functionality on low-power cores. This confirmed that the proposed DBSCAN clustering accelerator efficiently recognizes objects through LiDAR's point cloud without excessive power consumption, even in power-sensitive applications like autonomous robots.

The main contributions of this paper are as follows:

- The DBSCAN algorithm with grid for solid state LiDAR effectively saved computational burden.
- The DBSCAN accelerator significantly decreases computation time of clustering.
- The feasibility of the DBSCAN accelerator in autonomous robots was confirmed by connecting it to a low-power core, demonstrating effective object detection with low power consumption.

The paper is organized as follows: Section 2.1 discusses related work on LiDAR DBSCAN clustering, low-power object detection, and the DBSCAN clustering accelerator. Section 3 describes the architecture and operation of the designed DBSCAN clustering accelerator with a low-power core, with detailed explanations of the structure and operating principles of the included DBSCAN clustering accelerator. Section 4 presents the experimental setup and results for the accelerator. Finally, Section 5 concludes the paper, summarizing the research and contributions.

2. Related Work

2.1. DBSCAN LiDAR Clustering

Attempts to cluster LiDAR data began in 2008, when point cloud data from terrain features like mountains measured by LiDAR were leveraged to differentiate between ground

and off-ground [26]. The hierarchical and K-means clustering methods were employed for data mining, and experiments have confirmed the applicability of clustering algorithms to LiDAR's point cloud data. In 2013, researchers [27] compared different kinds of clustering algorithms applied on LiDAR's point cloud data for the first time. The study measured the similarity of clusters with adjusted rand index (ARI) compared to manually clustered data from an overhead view of LiDAR data and the time taken for density-based clustering and ordering points to identify the clustering structure (OPTIC) clustering algorithms. DBSCAN clustering results in a slightly better ARI than OPTIC clustering, verifying its suitability for LiDAR's point cloud. The following describes a study about DBSCAN clustering on LiDAR's point clouds. The study [28] involves automatic parameter tuning for DBSCAN clustering in autonomous driving scenarios, setting different *eps* and *MinPts* for objects at varying distances, achieving comparable accuracy to the original results. Additionally, the study [29] applies automatic *eps* estimation for DBSCAN segmentation of LiDAR data, employing the average of the maximum distances of k-nearest neighbors, and conducts experiments on mobile and airborne LiDAR data to improve accuracy. The experiments [29] resulted in accuracies of 75%, 74%, and 71%, which are higher than that of experiments with arbitrary parameters. The study in 2019 [30] organized effective DBSCAN clustering in two stages: the first stage involved ground plane fitting to obtain subclusters, then DBSCAN clustering was applied to these subclusters, efficiently avoiding oversegmentation problems. Furthermore, various studies attempt object detection on LiDAR's point cloud with the DBSCAN clustering algorithm [31–33]. Additionally, efforts for precise object detection with DBSCAN clustering include calibrating with camera sensors [34,35] and integrating deep learning [33,36]. Other clustering algorithms are also being adopted for object detection in LiDAR's point clouds across various studies [20,37].

2.2. DBSCAN Clustering Accelerator

Since the initial application of the DBSCAN method to LiDAR point cloud data in the 2013 study [26], the necessity to accelerate clustering speeds to enable the use of DBSCAN-based object detection in autonomous driving has emerged. One approach to enhance DBSCAN speed involves parallelizing the process. The study for parallelizing [38] adopted a disjoint-set data structure for parallelization of DBSCAN clustering, which improves memory efficiency and accelerates processing time up to 25.97 times. An advancement of this method, highly parallel DBSCAN, divides the area for local clustering and conducts parallel DBSCAN clustering considering the computation load of each area [39]. During the merging of labeled groups, re-labeling occurs at the boundaries.

Reducing the time complexity of the algorithm also contributes to an increase in the speed of the DBSCAN clustering. In 2013, researchers [40] drew a grid at *eps* intervals and incorporated the neighboring cells into a range query, applying a grid in partitioning based on 2D clustering. A study in 2020 [41] applies grid-based DBSCAN and a sliding window approach to large datasets, effectively reducing time with $O(n\sqrt{n})$ while maintaining accuracy similar to the original DBSCAN algorithm. 3D voxel-based clustering applies grid-based DBSCAN on 3D point clouds [42]. A study [43] adopts grid-based DBSCAN for parallel processing with a tree structure, enabling merge groups for various numbers of clusters, reducing the time complexity to $O(n \log n)$. For efficient DBSCAN clustering, the optimization of the time complexity of algorithms is being studied with various methods [44–46]. Table 1 organizes time complexity of each related work.

Other methods propose employing AI chips [47] to accelerate DBSCAN for large datasets. On such study [47] adopts a bitwise method for neighboring identification instead of compare-and-select operations, which are not familiar to AI chips, and two speculative execution strategies to eliminate data overhead. Evaluation of the study [47] shows 2.61 times speedup on NVIDIA RTX 3090. Similarly, ray tracing DBSCAN (RT-DBSCAN) leverages ray-tracing hardware for acceleration by translating fixed-radius nearest neighbor queries to ray-tracing queries [48].

A hardware accelerator is also an effective method for accelerating DBSCAN clustering. Two studies [49,50] implemented parallel DBSCAN on field programmable gate array (FPGA) to enhance clustering speed. One study [49] includes a processing elements (PE) array and collision check system, which is proper for FPGA. The other study [50] performs DBSCAN clustering on a multidimensional dataset and demonstrates a consistently brief execution time, regardless of the number of points. Additionally, studies [51,52] apply DBSCAN to millimeter wave (MMW) and radio frequency (RF) signals on FPGAs, enabling high-speed communications in noisy conditions. Another study [53] performs DBSCAN clustering on FPGAs for knowledge discovery of high-dimensional time series data and proposes the clustering system for high throughput and performance which supports fast machine learning.

Our previous research [54] also verified the operation of a DBSCAN accelerator designed with Verilog and implemented on an FPGA, which successfully accelerated the clustering and demonstrated the feasibility of a DBSCAN clustering accelerator for LiDAR's point clouds.

Table 1. Analysis of related works for efficient DBSCAN clustering in time complexity.

Ref.	Algorithm	Time Complexity	Features
KM Kumar et al., 2016 [44]	G-DBSCAN	$O(n + nd)$	A novel graph-based index structure method that groups data to accelerate the neighbor search operations.
Y Chen et al., 2021 [45]	BLOCK-DBSCAN	$O(n \log n)$	Approximation algorithm for large-scale data, judging whether two Inner core blocks are density reachable.
Y Wang et al., 2020 [29]	Practical Parallel DBSCAN	$O(n \log n)$	Grid-based DBSCAN for parallel processing, with tree structure enabling merge groups for a variety # of clusters
N Ohadi et al., 2020 [41]	SW-DBSCAN	$O(n\sqrt{n})$	Grid-based DBSCAN and a sliding window approach to large datasets
N Hanafi et al., 2022 [46]	OP-DBSCAN	$O(nk + nu)$	Using local samples and concept of operational dataset to find neighbors

d = # of dimensions of pattern, k = size of operation (OP), u = # of update operation.

2.3. Low-Power Object Detection

As research on object detection progresses, studies on low-power object detection have also commenced. The study [55] investigates real-time object detection methods for low-power embedded platforms and proposes approaches including finding suitable layers for the hardware, recognizing the limitations of processors and memory, identifying blocks with low computational complexity, and considering sparsification and quantization. Another study [56] suggests a high-power efficiency real-time object detection system. Optimization on the CNN algorithm and on the hardware platform achieves high accuracy and low energy consumption when implemented on FPGA.

Similarly, research into clustering for low-power applications has been conducted. The research [57] explored the implementation of clustering algorithms on low-power and high-performance devices, comparing the power efficiency with algorithms run on standard cores. While the clustering speed on low-power cores did not match that of standard cores, it demonstrated significantly better power efficiency.

2.4. Object Detection for Autonomous Driving with Various Sensors

Object detection for autonomous driving is conducted using various sensors, including cameras, LiDAR, and RADAR. These sensors are utilized independently or in combination through sensor fusion to perform object recognition. To achieve object detection for autonomous driving with camera sensors, one approach involves the use of fisheye lenses. While fisheye lenses offer the advantage of a wide field of view, they introduce distortions

that affect the accuracy of object recognition. Consequently, research has been conducted to correct these distortions, thereby enhancing the accuracy of object detection [58,59].

Another important sensor is the RADAR sensor using millimeter-wave (mmWave) technology. MmWave utilizes high-frequency radio waves to detect objects, providing longer detection ranges and strong resistance to environmental changes compared to other sensors, although it has the drawback of lower resolution. Object detection using mmWave is typically performed through sensor fusion with cameras, complementing the limitations of visual information provided by cameras. This sensor fusion approach enables more accurate and reliable object detection in autonomous driving systems, contributing to enhanced safety and efficiency [14,60,61].

3. Architecture

3.1. DBSCAN Clustering System with a Low-Power Core

Figure 2 illustrates the block diagram of the system. The proposed design is organized around the advanced high-performance bus (AHB) and the advanced peripheral bus (APB). Modules requiring high-speed bus interfaces for data transfer, such as the Cortex-M0, on-chip memory for code and data, and the DBSCAN clustering accelerator, are connected to the AHB. Peripheral modules that do not necessitate high-speed data communication, such as GPIO, universal asynchronous receiver transmitters (UART), and a timer, utilize the APB interface. The AHB to APB bridge facilitates communication between the two buses by changing bus the signal.

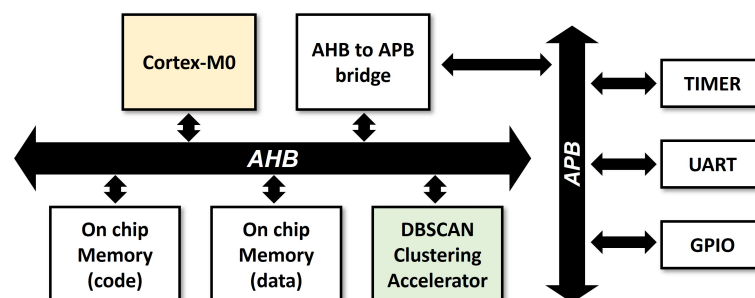


Figure 2. Architecture of DBSCAN clustering system with a low-power core.

The Cortex-M0, which is a low-power core, functions as the master of the AHB, accessing each module via addresses assigned to each peripheral and module, and reads and executes instructions fetched from on-chip memory, controlling the modules. The on-chip memory comprises two sections: one for storing the instruction codes necessary for program execution and another for storing data required during code execution on the core. The UART communicates with the PC, transmitting the point cloud dataset for clustering to register arrays on the accelerator. Once clustering is complete for all frames, the grouped data stored in register arrays are transmitted to the PC via the UART module. The GPIO module facilitates additional control based on the clustered group data by connecting to extra modules by supporting communication with other sensors or motors.

Figure 3 represents the flow of the DBSCAN clustering system. For executing the Cortex-M0, bit-stream compiled on Keil uVision 5 with C codes is stored on on-chip memory for code. Then x, y, z coordinates data and a list of pixels which is sorted in order to cluster them, defined as a seed-list are then transmitted to register arrays in a clustering accelerator from on-chip memory. With the start signal occurring after the save is finished, the clustering accelerator clusters the frame with the stored data in register memory arrays. When the clustering is finished, the accelerator transmits the group information resulting from the clustering accelerator to the PC via UART.

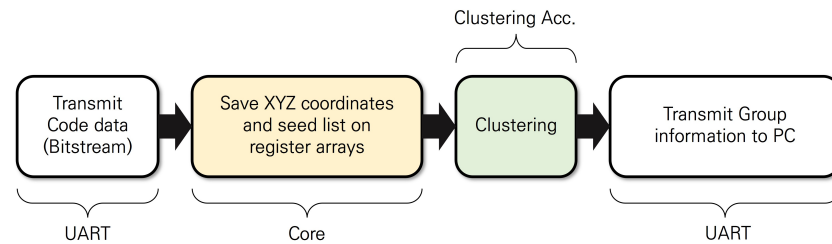


Figure 3. Flow of the DBSCAN clustering system.

3.2. DBSCAN Clustering Accelerator

3.2.1. Proposed DBSCAN Algorithm

The clustering algorithm adopted in the proposed DBSCAN clustering accelerator differs from the original DBSCAN algorithm. The original DBSCAN algorithm requires distance information between the core point and all other neighbor points for clustering. However, calculating distances for every point in numerous points during the clustering operation for each core point is computationally intensive and inefficient. The proposed method for the DBSCAN clustering applies the characteristics of LiDAR. LiDAR has fixed positions where light is emitted, and the number of points measured per frame is also fixed. These characteristics make it possible to impose grid-based DBSCAN on clustering LiDAR's cloud points, mapping the positions where light is emitted to a 2D grid.

Figure 4 is a 2D-mapped solid-state LiDAR point cloud. The proposed algorithm is designed based on the hardware specifications of LeddarTech's Pixell LiDAR. LeddarTech's Pixell covers approximately 180 degrees and emits light from 768 points, distributed across eight vertical and ninety-six horizontal positions, after which, it measures the distance. Thus, indices are mapped to 768 pixels for the proposed clustering algorithm. Figure 5 illustrates the horizontal angle of the field and positions.

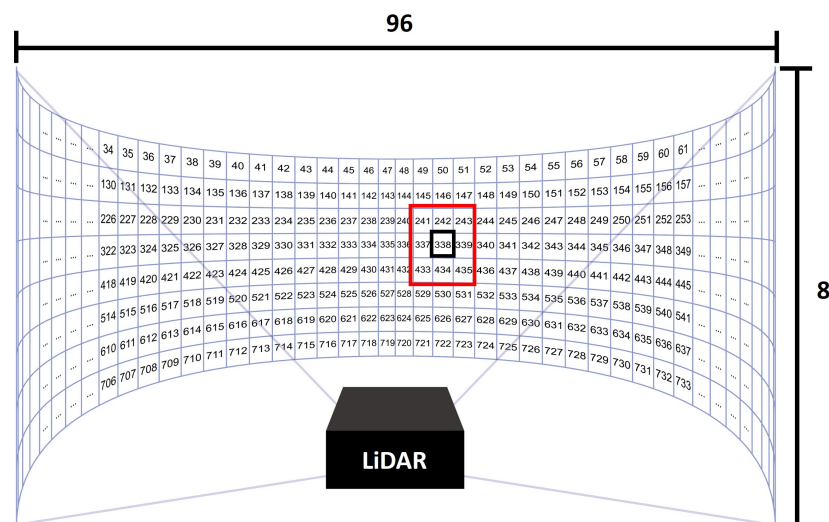


Figure 4. 2D-mapped solid-state LiDAR's point cloud.

Assigning indices reduces computational load since point information is stored and retrieved with index IDs. Furthermore, the accelerator minimizes the number of operations for clustering. Additionally, instead of measuring distances between a core point and all other points or having a pre-measured distance map for clustering, the proposed algorithm measures distances only for the eight pixels surrounding the core index pixel shown as Algorithms 1, 2 and Figure 4. In the figure, the core pixel is indicated by a black box, and the surrounding pixels are indicated by a red box. This method clusters by measuring distances only for these eight mapped indices, checking if the distances are smaller than ϵ . This approach reduces the time complexity from $O(n^2)$ in traditional DBSCAN to

$O(n)$ in the proposed method. A pixel becomes a core pixel either by being added to a group or selected as a seed pixel. Each pixel only becomes a core pixel once during the entire clustering process. The number of distance calculations when a pixel becomes a core is limited to a maximum of eight, resulting in a total of $8n$ times distance measuring operations per frame. In contrast, the original DBSCAN requires each of n pixels to measure distances n times, each with a more numerous computational load than the suggested algorithm. Therefore, the designed DBSCAN clustering accelerator aims to alleviate the number of operations performed with the proposed DBSCAN algorithm.

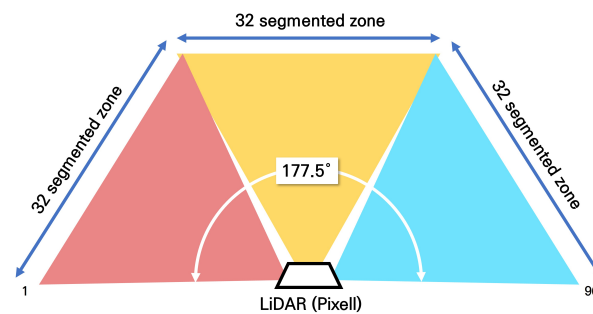


Figure 5. The horizontal angle of field and positions of LiDAR (Pixel).

Algorithm 1 Proposed grid-based DBSCAN algorithm

Require: eps , $MinPts$, $SeedList$

```

1:  $PosPixel \leftarrow \emptyset$ 
2: for seed point  $s \in SeedList$  do
3:   if  $AlreadyInGroup(s)$  then
4:     Continue
5:   end if
6:   while  $PosPixel \neq \emptyset$  do
7:     core point  $c = p \in PosPixel$ 
8:     if  $AlreadyInGroup(p)$  then
9:       Continue
10:    end if
11:     $Group \cup Clustering(c, eps, MinPts)$ 
12:     $PosPixel \cup Clustering(c, eps, MinPts)$ 
13:  end while
14:   $Grouplist \mathbf{append} Group$ 
15:   $PosPixel, Group \leftarrow \emptyset$ 
16:  core point  $c = seed\ point\ s$ 
17:   $Group \cup Clustering(c, eps, MinPts)$ 
18:   $PosPixel \cup Clustering(c, eps, MinPts)$ 
19: end for

```

Algorithm 2 $Clustering(c, eps, MinPts)$

Require: c , eps , $MinPts$

```

1: for  $i \in SurroundPixels$  do ▷ maximum 8 iterations
2:   if  $Dist(c, i) > eps$  then
3:      $Group_{3 \times 3} \leftarrow i$ 
4:   end if
5: end for
6: if  $\# of\ Group_{3 \times 3} > MinPts$  then
7:   return  $Group_{3 \times 3}$ 
8: else
9:   return  $\emptyset$ 
10: end if

```

3.2.2. Architecture

Figure 6 illustrates the structure of the DBSCAN clustering accelerator. The DBSCAN clustering accelerator consists of a grouping core and various register arrays such as pix use, seed list, xyz-list, and group-list. The pix-use array contains 768 1-bit signals, indicating whether a pixel index is included in a group (signaled by 1) or not yet grouped (signaled by 0) during clustering for a frame. The seed-list array and grouping core reference the pix-use array to determine if a seed pixel is already contained in a group or if a new pixel to be stored is already part of another group. The seed-list stores the index order of pixels to be clustered and xyz-list stores the x, y, z coordinate data corresponding to each index. Original DBSCAN clustering begins by selecting a core point from random points. However, as a random generator is difficult to implement in hardware environments, seed-list provides an index order of pixels. This order prioritizes pixels likely to form groups for each frame to facilitate clustering. The seed-list sequentially provides seed pixels upon request, checking via the pix-use array whether the seed pixel is part of a group before providing the seed pixel index to the grouping core. The xyz-list stores 24-bit data for each index, with x, y, z coordinate data occupying 8 bits each. Upon receiving a signal with a specific pixel index, the xyz-list provides the x, y, z coordinate data after one clock cycle. The group-list receives signals from the grouping core when a group is fully formed, and sequentially stores the indices of pixels included in the group. When clustering for all pixels is complete, the grouping core sends a frame-cplt signal, prompting the group-list to transmit the group information for the frame externally. Thus, each register array stores information about each pixel and connects to the grouping core, aiding in the clustering process.

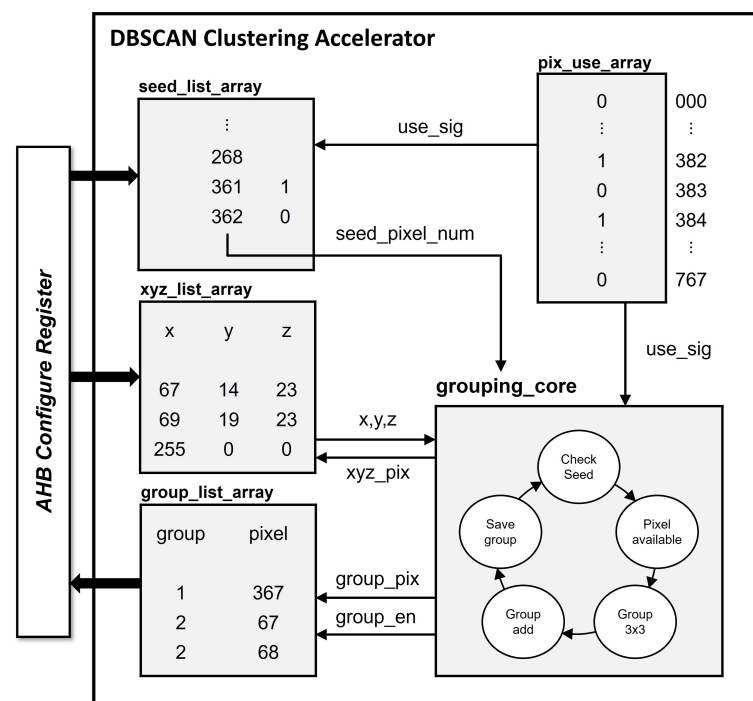


Figure 6. Architecture of the DBSCAN clustering accelerator.

The grouping core performs DBSCAN clustering based on LiDAR's point cloud data. The clustering process is divided into five main stages, shown as Figure 7, starting after the data are stored in the xyz-list and seed-list. These stages are seed-chk, pix-avl, group3x3, group-add, and save-group and the flow of the stages in the grouping core is illustrated in Figure 8. In the seed-chk stage, the grouping core sends a signal to the seed-list to get a seed pixel. If seed finding is complete, it goes to pix-avl stage. In the pix-avl stage, clustering determines which pixel is a core point, seed pixel or pos pixel. Pos-pixel comprises pixels included in the forming group. If the selected core pixel's x, y, z coordinates are not dummy

data, a calculating neighbor pixel is applied in stage group3x3. In group3x3, the distance is computed from the core pixel indicated as a red box in Figure 8 to up to eight surrounding pixels to determine if the number of points within the eps and compared with the number of pixels in group3x3, including the core pixel whether number meets or exceeds $MinPts$. If no group has formed, it goes to seed-chk and a seed-pixel is brought in as the new core pixel. If a group is forming, the group-add stage merges a newly formed small group to an existing group if one exists, or creates a new group if none exists. Once the group-add process is completed, it returns to the pix-avl stage and a pixel index from the pos-pixel group is designated as the core pixel. If no pos pixels remain for further core pixel evaluation after a group is formed, the process transitions to the save-group stage where the newly formed group is saved to the group-list. After the save-group is completed, the entire process is repeated until all pixels have undergone the grouping process. When clustering ends, a frame completion signal is sent to the group-list. Subsequently, the group-list is transmitted as per the core's request.

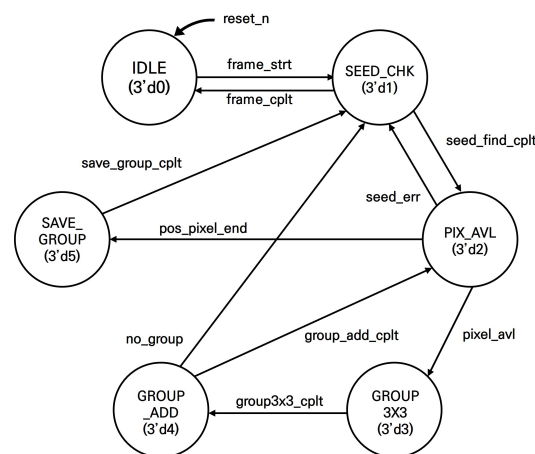


Figure 7. Finite state diagram of DBSCAN clustering accelerator.

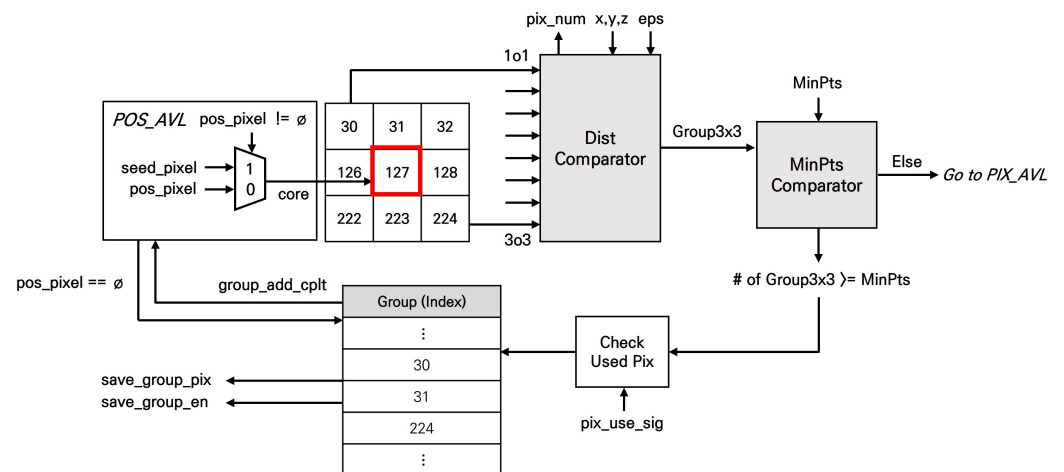


Figure 8. Flow of the grouping core.

4. Implementation and Analysis

4.1. Environment Setup

The dataset for validating the DBSCAN clustering accelerator is Pixset [62]. Released by LeddarTech in 2021, the Pixset dataset encompasses data from a combination of sensors, including cameras, LiDAR, RADAR, and IMUs. Each piece of data extracted from these sensors is timestamped with synchronized timing, which helps with object detection through sensor calibration. The Pixset dataset includes measurements from various urban environments such as highways, city streets, suburban areas, and boulevards, and also

covers a range of weather conditions, including cloudy, snowy and rainy at various times of day. Thus, the Pixset dataset provides point cloud data that closely reflect real-world driving environments and include realistically occurring obstacles.

The LiDAR equipment hired in the Pixset dataset [62] is LeddarTech's Pixell LiDAR, which refreshes frames at a rate of 20 Hz and measures up to 768 points per frame across a grid of ninety-six horizontal and eight vertical lines, with a field of view spanning 177.5 degrees horizontally and 16.0 degrees vertically. This wide coverage makes the Pixell suitable for experiments that cover large areas with a single LiDAR unit. The detection range of the Pixell is up to 56 m for an object which has 80% reflectivity, with a maximum detection range of 32 m for pedestrians. The proposed clustering accelerator is designed to perform object detection solely based on LiDAR point clouds. Additionally, for object detection on low-power cores, it is crucial to reduce computational load with a modest dataset. Pixell's relatively low number of 768 points per frame makes it well-suited for experiments involving accelerators connected to low-power cores. The designed accelerator is optimized for the size of the Pixell dataset, enhancing area efficiency.

The point cloud data from the Pixset dataset are not immediately suitable for the accelerator and need several modifications. First, dummy data padding is applied for indexing. The LiDAR data in the Pixset dataset consist of raw data measured by the light emitted and received by the LiDAR, not x , y , z coordinate data. As distance calculations in clustering relies on differences in x , y , z coordinates, the application programming interface (API) provided by LeddarTech is imported on raw data [62]. The API synchronizes data from multiple sensors via timestamps and extracts available data from raw data with the Pixset dataset. The API also provides a function to extract x , y , z coordinates from raw LiDAR data. However, the extracted coordinates do not cover indices for all pixels, meaning not all points in a frame are represented in the data. This is because points that are not measured are omitted from the frame. To proceed with the proposed clustering method, indices for each point in the dataset are necessary. Therefore, based on the extracted x , y , z coordinates, angles were measured with the specification of the angle of the field of the LiDAR and indices were assigned to each point. Indices that were not assigned are padded with designated dummy values, which does not affect clustering. Consequently, each frame contains x , y , z coordinates for 768 indexed points. The extracted x , y , z coordinates were quantized to be optimized in hardware environments. Since the Pixell LiDAR detects pedestrians up to 32 m away, points exceeding 30 m were designated as noise. Furthermore, as floating-point operations consume significant computational resources in hardware environments, x , y , z coordinates are quantized to 8-bit values, ranging from 0 to 255 for each x , y , and z . By applying two modifications of the dataset, the experiment is implemented with a lightweight dataset.

4.2. Experiments

The experiments were conducted in three scenarios. Initially, clustering is performed on a software program of an original DBSCAN algorithm and a software program of a proposed grid-based DBSCAN algorithm, written in python code, implemented on standard core with a quantized dataset. Next, Raspberry Pi 4 model B as a low-power device implements Python-coded software using version 3.12.2 of proposed algorithm on the quantized dataset. Lastly, the proposed DBSCAN clustering accelerator, which is connected to a low-power core, is designed with Verilog and implemented on an FPGA. The standard core applied in experiments to compare the results of the core that consumes normal power is an Intel's i7-12700 CPU, which operates on 2.1 GHz. The Raspberry pi 4 model B operates on 1.5 GHz and an embedded ARM Cortex-A72 microprocessor and results in efficient power consumption. The experiments compared the power consumption and processing time on clustering across three platforms. Groups formed through clustering were visually compared to confirm the successful implementation and operation of the algorithm.

The experimental setup for validating the proposed accelerator and details about the FPGA board are depicted in Figure 9. The FPGA board is a custom-designed board for

verifying the accelerator. The FPGA includes a low-power core, which is the Cortex-M0, and a proposed accelerator. The main FPGA is Altera's MAX 10 (10M50SCE114C8G). The FPGA board has a UART port to receive the bit-stream containing the code to be executed on the core and to transmit the clustering results, a JTAG for uploading the proposed design to the MAX 10, and GPIOs to connect various sensors on Cortex-M0 for the future work. The FPGA receives the code to be executed and the dataset of the frame via UART, performs clustering via a reset button, and then sends the clustering results back through UART. The proposed design with the Max 10 FPGA operates on 50 MHz, which is relatively slow and consumes less power.

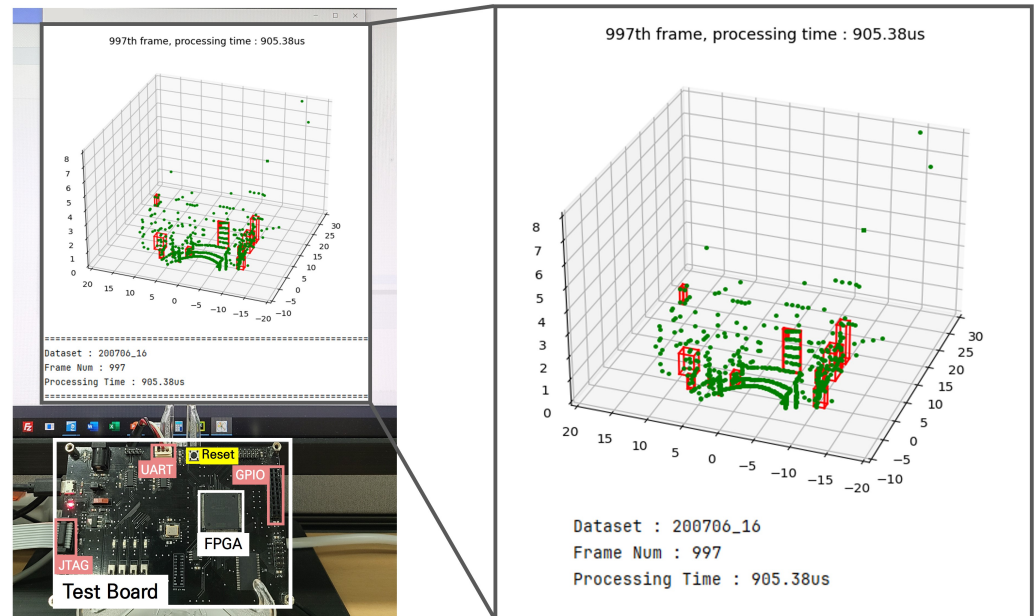


Figure 9. Experiment environment for implementation of proposed accelerator and a clustering result of LiDAR's point cloud.

Table 2 represents the hardware resources of the proposed accelerator with Cortex-M0 and only one accelerator. The hardware resources are analyzed with the number of look-up tables (LUTs), Registers, and block random access memories (BRAMs). The hardware implementation uses 17,648 LUTs, 13,687 Registers and 34,816 BRAMs when only DBSCAN accelerator is synthesized, utilizing and 24,806 LUTs, 16,170 Registers, 296,996 BRAMs when the proposed accelerator with Cortex-M0 is synthesized. Each employs 61% and 45% logic elements, and 18% and 2% BRAMs of MAX 10, demonstrating the proposed accelerator with a low-power core runs on MAX 10 FPGA without hardware overhead.

Table 2. The hardware resources of the Acc. with Cortex-M0 and only Acc.

	LUTs	Registers	BRAMs	Logic Elements
Max 10	-	-	1,677,321	49,760
Acc. with Cortex-M0.	24,806	16,170	296,996 (18%)	30,533 (61%)
Only Acc.	17,648	13,687	34,816 (2%)	22,972 (45%)

The experiments were conducted on a total of 3806 frames recorded in various conditions such as downtown, suburban, and main roads during clear, cloudy, rainy weather, at both day and night. Table 3 shows the information of each dataset. The proposed accelerator was tested on 20 frames from each dataset, a total of 100 frames.

Table 4 measures the average clustering processing time per frame on different platforms. For the original DBSCAN algorithm clustered on the i7-12700, the average processing time per frame was recorded at 224.48 ms, while for the proposed grid-based algorithm, the processing time was reduced to 8.45 ms, demonstrating that quantization, which optimizes the clustering process for LiDAR, effectively shortens processing times. Clustering on the Raspberry Pi 4, which adopts quantized data, resulted in a comparatively slower average processing time of 45.69 ms relative to other platforms. On the other hand, the proposed accelerator recorded an average processing time of just 0.886 ms per frame. Thus, the proposed algorithm speeds up the clustering process by 21.88 times compared to the original DBSCAN algorithm, and the proposed accelerator speeds up the clustering process by 51.57 and 9.54 times each compared to the low-power device, Raspberry Pi 4 and a standard core i7-12700, which used quantized data, showing significant improvements on processing time.

Table 3. Specification of each dataset in Pixset.

	Dataset (a)	Dataset (b)	Dataset (c)	Dataset (d)	Dataset (e)
Time	day	day	night	day	day
Weather	clouds	clouds	clouds	clouds	rain
Location	downtown	suburban	boulevard	highway	boulevard
# of frames	2929	290	275	190	122

Table 4. Average Processing times per frame on each platform.

Platform	Dataset (a)	Dataset (b)	Dataset (c)	Dataset (d)	Dataset (e)	Average
Original DBSCAN, i7-12700	220.35	253.8	220.4	245.8	230.07	224.48
Proposed algorithm, i7-12700	7.87	14.91	8.29	7.24	7.24	8.45
Raspberry Pi 4	43.87	63.85	46.27	44.74	44.74	45.69
Proposed Accelerator *	0.89	0.88	0.89	0.87	0.9	0.886

All results are expressed in milliseconds [ms]. * Proposed accelerator is tested by 20 frames on each dataset.

Table 5 presents a visual comparison of LiDAR's clustering results involving the original DBSCAN algorithm, proposed algorithm, and the proposed accelerator, alongside the raw pictures from a camera sensor included in the Pixset on same frames. Clustering of the original DBSCAN algorithm and proposed grid-based DBSCAN algorithm was performed on a standard core through Python, while the proposed accelerator employed a low-power core. Through the visual comparison of clustering results for each frame, the clustering involving the proposed algorithm and the accelerator proceeded correctly without significant differences compared to the clustering involving the original DBSCAN algorithm in visual.

To compare the power consumption of the clustering accelerator with a low-power core implemented on MAX 10, the power consumed during clustering on each platform was analyzed. Power consumption is calculated based on the formula from [57]. The formula produces the execution time (T) and thermal design power (TDP) to obtain power consumption information on the system for executing clustering. More precisely,

$$H = T \cdot TDP = \frac{[second]}{[second]} \cdot \frac{[Joule]}{[second]}$$

(1)

Table 5. LiDAR’s point cloud clustering results compared to raw pictures from a camera sensor on same frames.

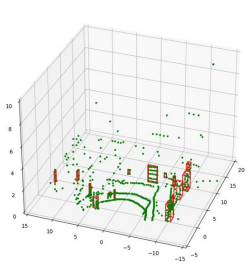
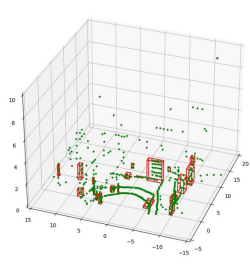
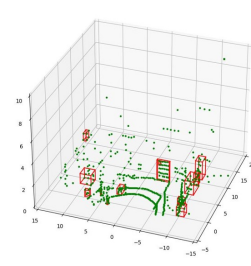
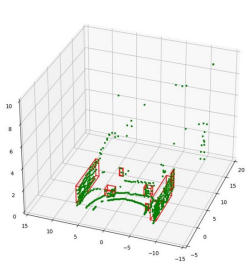
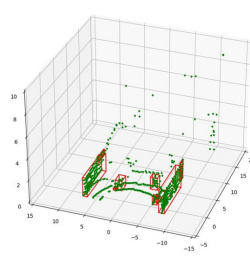
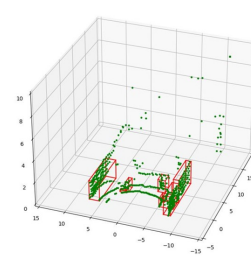
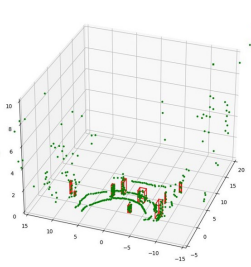
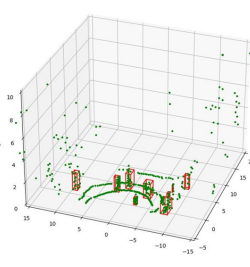
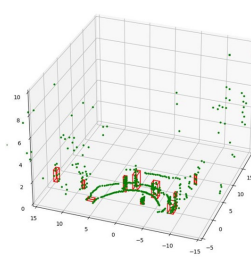
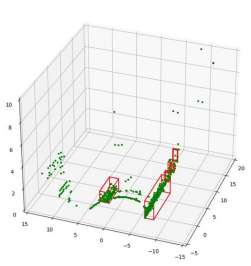
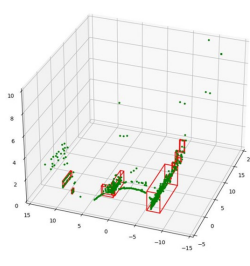
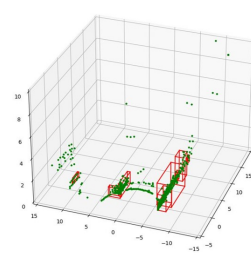
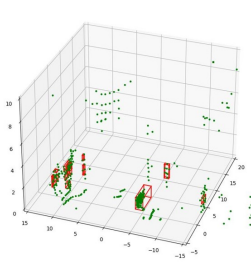
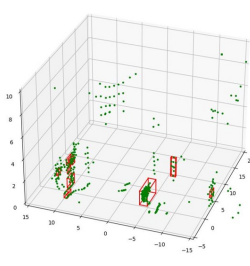
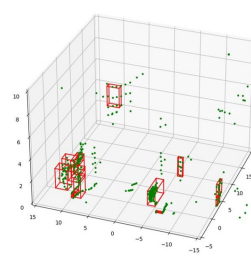
Dataset	Raw Picture	Original DBSCAN SW	Proposed Algorithm SW	Proposed Acc.
Dataset (a), frame 997				
Dataset (b), frame 24				
Dataset (c), frame 166				
Dataset (d), frame 41				
Dataset (e), frame 37				

Table 6 presents the TDP and power consumption of the system during clustering of the Pixset dataset for three different clustering platforms. The TDP for the i7-12700 is based on processor base power data provided by Intel, while for the Raspberry Pi 4, TDP is based on idle power consumption from power consumption benchmarks. The power consumed by the FPGA is based on power estimation performed through Quartus for a MAX 10 FPGA running at 50 MHz. The i7-12700's TDP is 65 W, which is common to CPU for a desktop. And Raspberry Pi 4 and FPGA's TDP are each 2.7 W and 1.035 W, which indicates the feature of a low-power device. The H , computed for power consumption, of i7-12700 is 0.54925 J, which is higher, despite the fast speed of clustering. The Raspberry Pi 4's H is computed as 0.12336 J, which is low, despite Raspberry Pi 4's low speed of clustering. However, the FPGA's H is recorded as 0.00092 J, which is very low value compared to Raspberry Pi 4's H , which is also a low-power device, showing a 99% reduction rate on power consumption. As a result, the FPGA, on which the proposed accelerator is implemented, effectively lessens the power consumption compared to other platforms.

Table 6. Power consumption of clustering on each platforms.

	Quantized, i7-12700	Raspberry Pi 4	FPGA
TDP (W)	65	2.7	1.035
$H(J)$	0.54925	0.12336	0.00092

5. Conclusions

Research on autonomous robots has continued to progress and commercial attempts at developing object detection technology, essential for autonomous navigation, have been made. However, object detection in autonomous robots equipped with batteries considers power efficiency and is thus capable of operating on low-power cores. To execute object detection in autonomous robots, we propose a grid-based DBSCAN clustering accelerator for LiDAR's point cloud.

The proposed accelerator employs an algorithm adapted to the characteristics of LiDAR, using grid-based DBSCAN clustering. The algorithm performs 2D mapping based on the emission points of LiDAR light and assigns indices to each location. By bounding distance calculations to adjacent indices, the proposed algorithm decreases the algorithm's time complexity from $O(n^2)$ to $O(n)$. The accelerator is connected to the Cortex-M0 through AHB, allowing operation within a low-power environment.

Experiments of clustering were executed on quantized data with original DBSCAN algorithm through the i7-12700 and the proposed DBSCAN algorithm through i7-12700, Raspberry Pi 4, and the proposed accelerator with a low-power core implemented on an FPGA, with outcomes compared across these platforms. The proposed algorithm revealed a reduction in time due to streamlined computations, and the proposed accelerator exhibited faster clustering speeds than other platforms. Power efficiency assessments further indicated that the proposed accelerator with a low-power core surpassed the Raspberry Pi 4 in performance. Visually comparing the clustering results across platforms confirmed that the proposed algorithm successfully conducts clustering and that the results from the proposed accelerator are consistent with those from other platforms.

This research verifies the feasibility of the a proposed DBSCAN clustering accelerator with a low-power core and the proposed accelerator with a general purpose core has scalability to connect other various sensors, especially a LiDAR. Attaching the LiDAR on the core allows real-time object detection with the proposed accelerator, and diverse applications for autonomous robots. To be applied in diverse applications, verifying the operation with various LiDAR specifications is essential to enhance the algorithm.

Author Contributions: Conceptualization, S.L. and S.E.L.; methodology, S.L., S.A. and S.E.L.; software, S.L. and S.A.; hardware, S.L.; validation, S.L., S.A., J.K., H.N. and R.K.; formal analysis, S.L. and J.K.; investigation, S.L. and J.K.; data curation, S.L., H.N. and J.P.; writing—original draft preparation, S.L.; writing—review and editing, S.L., R.K., J.P. and S.E.L.; visualization, S.L. and R.K.; supervision, S.E.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-RS-2022-00156295) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

Data Availability Statement: The data presented in this study are available in this article.

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

References

- Sharma, T.; Debaque, B.; Duclos, N.; Chehri, A.; Kinder, B.; Fortier, P. Deep Learning-Based Object Detection and Scene Perception under Bad Weather Conditions. *Electronics* **2022**, *11*, 563. [\[CrossRef\]](#)
- Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [\[CrossRef\]](#)
- Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* **2020**, *8*, 58443–58469. [\[CrossRef\]](#)
- Shin, J.Y.; Ho Lee, S.; Go, K.; Kim, S.; Lee, S.E. AI Processor based Data Correction for Enhancing Accuracy of Ultrasonic Sensor. In Proceedings of the 2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hangzhou, China, 11–13 June 2023; pp. 1–4. [\[CrossRef\]](#)
- Jeong, Y.; Jeong, W.S.; Shin, J.Y.; Lee, S.E. The Design of Embedded Fuzzy Logic Controller for Autonomous Mobile Robots. In Proceedings of the 2023 20th International SoC Design Conference (ISOCC), Jeju, Republic of Korea, 25–28 October 2023; pp. 145–146. [\[CrossRef\]](#)
- Jeong, Y.W.; Go, K.H.; Lee, S.E. Robot-on-Chip: Computing on a Single Chip for an Autonomous Robot. In Proceedings of the 2022 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 7–9 January 2022; pp. 1–3. [\[CrossRef\]](#)
- Shnaps, I.; Rimón, E. Online Coverage of Planar Environments by a Battery Powered Autonomous Mobile Robot. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 425–436. [\[CrossRef\]](#)
- Zou, B.; Xu, X.; De Koster, R. Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *Eur. J. Oper. Res.* **2018**, *267*, 733–753. [\[CrossRef\]](#)
- Jiang, P.; Ergu, D.; Liu, F.; Cai, Y.; Ma, B. A Review of Yolo Algorithm Developments. *Procedia Comput. Sci.* **2022**, *199*, 1066–1073. [\[CrossRef\]](#)
- He, K.; Gkioxari, G.; Dollar, P.; Girshick, R. Mask R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017.
- Park, J.; Shin, J.; Kim, R.; An, S.; Lee, S.; Kim, J.; Oh, J.; Jeong, Y.; Kim, S.; Jeong, Y.R.; et al. Accelerating Strawberry Ripeness Classification Using a Convolution-Based Feature Extractor along with an Edge AI Processor. *Electronics* **2024**, *13*, 344. [\[CrossRef\]](#)
- Kim, S.; Park, J.; Jeong, Y.; Lee, S.E. Intelligent Monitoring System with Privacy Preservation Based on Edge AI. *Micromachines* **2023**, *14*, 1749. [\[CrossRef\]](#)
- Mao, J.; Shi, S.; Wang, X.; Li, H. 3D object detection for autonomous driving: A comprehensive survey. *Int. J. Comput. Vis.* **2023**, *131*, 1909–1963. [\[CrossRef\]](#)
- Wei, Z.; Zhang, F.; Chang, S.; Liu, Y.; Wu, H.; Feng, Z. MmWave Radar and Vision Fusion for Object Detection in Autonomous Driving: A Review. *Sensors* **2022**, *22*, 2542. [\[CrossRef\]](#)
- Nabati, R.; Qi, H. RRPNet: Radar Region Proposal Network for Object Detection in Autonomous Vehicles. In Proceedings of the 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 22–25 September 2019; pp. 3093–3097. [\[CrossRef\]](#)
- Wang, Y.; Jiang, Z.; Li, Y.; Hwang, J.N.; Xing, G.; Liu, H. RODNet: A Real-Time Radar Object Detection Network Cross-Supervised by Camera-Radar Fused Object 3D Localization. *IEEE J. Sel. Top. Signal Process.* **2021**, *15*, 954–967. [\[CrossRef\]](#)
- Pang, S.; Morris, D.; Radha, H. CLOCs: Camera-LiDAR Object Candidates Fusion for 3D Object Detection. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 10386–10393. [\[CrossRef\]](#)
- Liang, Z.; Huang, Y. Survey on deep learning-based 3D object detection in autonomous driving. *Trans. Inst. Meas. Control* **2023**, *45*, 761–776. [\[CrossRef\]](#)
- Jin, X.; Yang, H.; He, X.; Liu, G.; Yan, Z.; Wang, Q. Robust LiDAR-based vehicle detection for on-road autonomous driving. *Remote Sens.* **2023**, *15*, 3160. [\[CrossRef\]](#)
- Adnan, M.; Slavic, G.; Martin Gomez, D.; Marcenaro, L.; Regazzoni, C. Systematic and comprehensive review of clustering and multi-target tracking techniques for LiDAR point clouds in autonomous driving applications. *Sensors* **2023**, *23*, 6119. [\[CrossRef\]](#)
- Kodinariya, T.M.; Makwana, P.R. Review on determining number of Cluster in K-Means Clustering. *Int. J.* **2013**, *1*, 90–95.

22. Nielsen, F.; Nielsen, F. Hierarchical clustering. *Introduction to HPC with MPI for Data Science*; Springer: Cham, Switzerland, 2016; pp. 195–211.
23. Kriegel, H.P.; Kröger, P.; Sander, J.; Zimek, A. Density-based clustering. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2011**, *1*, 231–240. [\[CrossRef\]](#)
24. Bushra, A.A.; Yi, G. Comparative analysis review of pioneering DBSCAN and successive density-based clustering algorithms. *IEEE Access* **2021**, *9*, 87918–87935. [\[CrossRef\]](#)
25. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the KDD, Portland, OR, USA, 2–4 August 1996; Volume 96, pp. 226–231.
26. Chehata, N.; David, N.; Bretar, F. LIDAR data classification using hierarchical K-means clustering. In *ISPRS Congress Beijing 2008*; Citeseer: University Park, PA, USA, 2008; Volume 37, pp. 325–330.
27. Ghosh, S.; Lohani, B. Mining lidar data with spatial clustering algorithms. *Int. J. Remote Sens.* **2013**, *34*, 5119–5135. [\[CrossRef\]](#)
28. Yabroudi, M.E.; Awedat, K.; Chabaan, R.C.; Abudayyeh, O.; Abdel-Qader, I. Adaptive DBSCAN LiDAR Point Cloud Clustering For Autonomous Driving Applications. In Proceedings of the 2022 IEEE International Conference on Electro Information Technology (eIT), Mankato, MN, USA, 19–21 May 2022; pp. 221–224. [\[CrossRef\]](#)
29. Wang, C.; Ji, M.; Wang, J.; Wen, W.; Li, T.; Sun, Y. An Improved DBSCAN Method for LiDAR Data Segmentation with Automatic Eps Estimation. *Sensors* **2019**, *19*, 172. [\[CrossRef\]](#)
30. Zhang, C.; Wang, S.; Yu, B.; Li, B.; Zhu, H. A Two-Stage Adaptive Clustering Approach for 3D Point Clouds. In Proceedings of the 2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), Nagoya, Japan, 13–15 July 2019; pp. 11–16.
31. Wu, J.; Xu, H.; Zheng, J.; Zhao, J. Automatic Vehicle Detection with Roadside LiDAR Data Under Rainy and Snowy Conditions. *IEEE Intell. Transp. Syst. Mag.* **2021**, *13*, 197–209. [\[CrossRef\]](#)
32. Cao, M.; Wang, J. Obstacle detection for autonomous driving vehicles with multi-lidar sensor fusion. *J. Dyn. Syst. Meas. Control* **2020**, *142*, 021007. [\[CrossRef\]](#)
33. Nguyen, H.T.; Lee, E.H.; Bae, C.H.; Lee, S. Multiple Object Detection Based on Clustering and Deep Learning Methods. *Sensors* **2020**, *20*, 4424. [\[CrossRef\]](#)
34. Verucchi, M.; Bartoli, L.; Bagni, F.; Gatti, F.; Burgio, P.; Bertogna, M. Real-Time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars. In Proceedings of the 2020 Fourth IEEE International Conference on Robotic Computing (IRC), Taichung, Taiwan, 9–11 November 2020; pp. 398–405.
35. Tamayo, W.C.; Chelbi, N.E.; Gingras, D.; Faulconnier, F. Improving Object Distance Estimation in Automated Driving Systems Using Camera Images, LiDAR Point Clouds and Hierarchical Clustering. In Proceedings of the 2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops), Nagoya, Japan, 11–17 July 2021; pp. 299–305.
36. Wong, K.; Wang, S.; Ren, M.; Liang, M.; Urtasun, R. Identifying unknown instances for autonomous driving. In Proceedings of the Conference on Robot Learning, Virtual, 16–18 November 2020; pp. 384–393.
37. Gao, F.; Li, C.; Zhang, B. A dynamic clustering algorithm for lidar obstacle detection of autonomous driving system. *IEEE Sens. J.* **2021**, *21*, 25922–25930. [\[CrossRef\]](#)
38. Patwary, M.M.A.; Palsetia, D.; Agrawal, A.; Liao, W.K.; Manne, F.; Choudhary, A. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In Proceedings of the SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 10–16 November 2012; pp. 1–11.
39. Götz, M.; Bodenstern, C.; Riedel, M. HPDBSCAN: Highly parallel DBSCAN. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, Austin, TX, USA, 15–20 November 2015; pp. 1–10.
40. Gunawan, A.; de Berg, M. A Faster Algorithm for DBSCAN. Master's Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2013.
41. Ohadi, N.; Kamandi, A.; Shabankhah, M.; Fatemi, S.M.; Hosseini, S.M.; Mahmoudi, A. Sw-dbscan: A grid-based dbscan algorithm for large datasets. In Proceedings of the 2020 6th International Conference on Web Research (ICWR), Tehran, Iran, 22–23 April 2020; pp. 139–145.
42. Hrutka, B.; Siki, Z.; Takács, B. Voxel-based point cloud segmentation and building detection. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2022**, *48*, 209–215. [\[CrossRef\]](#)
43. Wang, Y.; Gu, Y.; Shun, J. Theoretically-efficient and practical parallel DBSCAN. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 2555–2571.
44. Mahesh Kumar, K.; Rama Mohan Reddy, A. A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognit.* **2016**, *58*, 39–48. [\[CrossRef\]](#)
45. Chen, Y.; Zhou, L.; Bouguila, N.; Wang, C.; Chen, Y.; Du, J. BLOCK-DBSCAN: Fast clustering for large scale data. *Pattern Recognit.* **2021**, *109*, 107624. [\[CrossRef\]](#)
46. Hanafi, N.; Saadatfar, H. A fast DBSCAN algorithm for big data based on efficient density calculation. *Expert Syst. Appl.* **2022**, *203*, 117501. [\[CrossRef\]](#)
47. Ji, Z.; Wang, C.L. Accelerating DBSCAN algorithm with AI chips for large datasets. In Proceedings of the 50th International Conference on Parallel Processing, Lemont, IL, USA, 9–12 August 2021; pp. 1–11.
48. Nagarajan, V.; Kulkarni, M. RT-DBSCAN: Accelerating DBSCAN using Ray Tracing Hardware. In Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), St. Petersburg, FL, USA, 15–19 May 2023; pp. 963–973.
49. Shi, Q.Y.S.; Wang, Q. FPGA based accelerator for parallel DBSCAN algorithm. *Comput. Model. New Technol.* **2014**, *18*, 135–142.

50. Scicluna, N.; Bouganis, C.S. ARC 2014: A multidimensional FPGA-based parallel DBSCAN architecture. *ACM Trans. Reconfigurable Technol. Syst. (TRETs)* **2015**, *9*, 1–15. [\[CrossRef\]](#)
51. Xu, W.; Li, P.; Zou, X.; Zhong, N.; Pan, W.; Yan, L. Proposal and FPGA implementation of DBSCAN clustering nonlinear detector for MMW RoF system. In Proceedings of the 2022 IEEE International Topical Meeting on Microwave Photonics (MWP), Orlando, FL, USA, 3–7 October 2022; pp. 1–4.
52. Gavin, B.; Deng, T.; Ball, E. Low Area and Low Power FPGA Implementation of a DBSCAN-Based RF Modulation Classifier. *IEEE Open J. Comput. Soc.* **2024**, *5*, 50–61. [\[CrossRef\]](#)
53. Porcello, J.C. Density-Based Clustering for Knowledge Discovery of High-Dimensional Time Series Data using FPGAs. In Proceedings of the 2021 IEEE Aerospace Conference (50100), Big Sky, MT, USA, 6–13 March 2021; pp. 1–7.
54. Lee, S.; An, S.; Kim, R.; Oh, J.; Lee, S.E. Point Cloud Clustering System with DBSCAN Algorithm for Low-Resolution LiDAR. In Proceedings of the 2024 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 6–8 January 2024; pp. 1–2. [\[CrossRef\]](#)
55. Jose, G.; Kumar, A.; Kruthiventi SS, S.; Saha, S.; Muralidhara, H. Real-time object detection on low power embedded platforms. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, Seoul, Republic of Korea, 27–28 October 2019.
56. Yu, J.; Guo, K.; Hu, Y.; Ning, X.; Qiu, J.; Mao, H.; Yao, S.; Tang, T.; Li, B.; Wang, Y.; et al. Real-time object detection towards high power efficiency. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 704–708. [\[CrossRef\]](#)
57. Lapegna, M.; Balzano, W.; Meyer, N.; Romano, D. Clustering Algorithms on Low-Power and High-Performance Devices for Edge Computing Environments. *Sensors* **2021**, *21*, 5395. [\[CrossRef\]](#)
58. Rashed, H.; Mohamed, E.; Sistu, G.; Kumar, V.R.; Eising, C.; El-Sallab, A.; Yogamani, S. FisheyeYOLO: Object detection on fisheye cameras for autonomous driving. In Proceedings of the Machine Learning for Autonomous Driving NeurIPS 2020 Virtual Workshop, Virtual, 11 December 2020; Volume 11, p. 1.
59. Yahiaoui, M.; Rashed, H.; Mariotti, L.; Sistu, G.; Clancy, I.; Yahiaoui, L.; Kumar, V.R.; Yogamani, S. Fisheyemodnet: Moving object detection on surround-view cameras for autonomous driving. *arXiv* **2019**, arXiv:1908.11789.
60. Song, Y.; Xie, Z.; Wang, X.; Zou, Y. MS-YOLO: Object detection based on YOLOv5 optimized fusion millimeter-wave radar and machine vision. *IEEE Sens. J.* **2022**, *22*, 15435–15447. [\[CrossRef\]](#)
61. Huang, X.; Tsoi, J.K.P.; Patel, N. mmWave Radar Sensors Fusion for Indoor Object Detection and Tracking. *Electronics* **2022**, *11*, 2209. [\[CrossRef\]](#)
62. Déziel, J.; Merriaux, P.; Tremblay, F.; Lessard, D.; Plourde, D.; Stanguennec, J.; Goulet, P.; Olivier, P. PixSet: An Opportunity for 3D Computer Vision to Go Beyond Point Clouds With a Full-Waveform LiDAR Dataset. In Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 19–22 September 2021; pp. 2987–2993. [\[CrossRef\]](#)

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