sensors

Article

Development of a GPU-Accelerated NDT Localization Algorithm for GNSS-Denied Urban Areas

Keon Woo Jang , Woo Jae Jeong and Yeonsik Kang *

Department of Automotive Engineering, Kookmin University, 77 Jeongneung-ro, Seongbuk-gu, Seoul 02707, Korea; rjsdn8769@kookmin.ac.kr (K.W.J.); wjddnwo95@kookmin.ac.kr (W.J.J.)
* Correspondence: ykang@kookmin.ac.kr; Tel.: +82-2-910-4671

Abstract: There are numerous global navigation satellite system-denied regions in urban areas, where the localization of autonomous driving remains a challenge. To address this problem, a high-resolution light detection and ranging (LiDAR) sensor was recently developed. Various methods have been proposed to improve the accuracy of localization using precise distance measurements derived from LiDAR sensors. This study proposes an algorithm to accelerate the computational speed of LiDAR localization while maintaining the original accuracy of lightweight map-matching algorithms. To this end, first, a point cloud map was transformed into a normal distribution (ND) map. During this process, vector-based normal distribution transform, suitable for graphics processing unit (GPU) parallel processing, was used. In this study, we introduce an algorithm that enabled GPU parallel processing of an existing ND map-matching process. The performance of the proposed algorithm was verified using an open dataset and simulations. To verify the practical performance of the proposed algorithm, the real-time serial and parallel processing performances of the localization were compared using high-performance and embedded computers, respectively. The distance root-mean-square error and computational time of the proposed algorithm were compared. The algorithm increased the computational speed of the embedded computer almost 100-fold while maintaining high localization precision.

Keywords: autonomous vehicle; NDT; localization; ROS; 3D LiDAR; GPGPU

check for updates

Citation: Jang, K.W.; Jeong, W.J.; Kang, Y. Development of a GPU-Accelerated NDT Localization Algorithm for GNSS-Denied Urban Areas. *Sensors* **2022**, *22*, 1913. https://doi.org/10.3390/s22051913

Academic Editors: Shiyang Tang, Yan Huang, Zhanye Chen and Ping Guo

Received: 19 January 2022
Accepted: 26 February 2022
Published: 1 March 2022

1. Introduction

Since the mid-1980s, remarkable research efforts and investments have been devoted to developing autonomous driving technology [1,2]. Automotive companies have dedicated remarkable efforts to the commercialization of this technology, which requires near-perfect safety conditions [3–5]. Therefore, precise real-time perception of the surrounding environment using various sensors, such as a global navigation satellite system (GNSS), cameras, light detection and ranging (LiDAR) equipment, radio detection and ranging (RADAR) technology, and the inertial measurement unit (IMU), is essential [6]. Diverse types of information are required for safe autonomous driving, including the location of the ego vehicle, road environment, and spatial relationship between the surrounding objects. The techniques used for extracting such information from the environment are interdependent. In addition, as most perception methods are performed under the assumption that the pose of the ego vehicle is accurately known, inaccurate pose information may result in the deterioration of the performance of the autonomous driving system. Although accurate information regarding the position of a vehicle can be obtained using GNSS technology [7–10], the localization error recommended by ISO 17572 (within 25 cm) cannot be guaranteed in all driving areas [11,12]. In particular, inaccurate GNSS location measurement occurs in GNSS-denied areas, such as urban canyons, overpasses, tunnels, and underpasses, where large objects obscure the sensor receiver [13]. To address this, the use of multiple sensors that can recognize the surrounding environment of a vehicle, rather than the singular use of GNSS technology, is essential for seamless localization [14–16].

Cameras have been widely used as sensors in autonomous vehicles [17–20], and numerous studies have investigated the efficacy of deep learning techniques for processing a large amount of image data. However, as an image captured by a camera records the environment in a two-dimensional (2D) aspect, it cannot be used to accurately estimate the three-dimensional (3D) environment. Cameras are therefore not suitable for establishing precise localization for autonomous driving. Conversely, LiDAR has attracted attention as a sensor for autonomous vehicles because of its ability to acquire 3D distance information from the environment with error rates within a centimeter. Another localization method based on simultaneous localization and mapping [21–23] has attracted attention because of its ability to simultaneously estimate the position of a vehicle and create a relevant map. However, the error accumulation that occurs during the estimation process may lead to distortion of the generated map [24]. Although this error accumulation can be corrected using a factor graph-based loop closure [25–27], this closure may lead to an abrupt correction in the position of the vehicle, which can result in a dangerous change in the route of an autonomous vehicle in a complex urban environment. In this study, we assumed that a precise map already existed and could be used for the localization of autonomous vehicles via a comparison with current LiDAR measurements [28–30].

Recent developments in LiDAR with improved spatial resolution include the 128-channel 3D LiDAR, which can measure more than 400,000 points in a single scan at an update rate of 10–20 Hz. To use such high-resolution LiDAR measurements for localization, a large point cloud must be rapidly processed to ensure precise control of the autonomous vehicle [31]. The normal distribution transform (NDT), a technique that represents a point cloud with a normal distribution, can enable a more compact spatial representation of a point cloud [32]. Therefore, this study used a vector-based normal distribution (ND) map [33]. Although the ND map for localization was represented compactly, the LiDAR still produced numerous measurements, which had to be rapidly and efficiently processed to enable real-time localization. To effectively accelerate the computational speed, this study utilized graphics processing unit (GPU) parallel processing.

The main contributions of this paper can be briefly summarized as follows:

- The proposal of a real-time localization algorithm applying GPU parallel processing to a vector-based normal distribution transform;
- Verification of the algorithm using both the nuScenes dataset (Motional, Boston, Massachusetts, United States) and the CarMaker simulation (IPG Automotive GmbH, Karlsruhe, Baden-Württemberg, Germany) to confirm that the algorithm could operate in a real urban environment and that continuous real-time localization was possible.

The rest of this paper is organized as follows. In Section 2, a method for converting a 3D point cloud map into a vector-based ND map and an efficient map-matching algorithm using GPU parallel processing for localization are described. The performance of the proposed algorithm is verified using an open dataset and simulation. The details of the verification method are described in Section 3. The performance results of the algorithm are discussed in Section 4, and the conclusion of this study is presented in Section 5.

2. Methods

Figure 1 demonstrates a flowchart of the proposed algorithm. The red area in the figure shows the ND map generation process, which utilized measurements from three sensors (LiDAR, GNSS, and IMU) to create the map, based on universal transverse mercator (UTM) coordinates. The 3D point cloud mapping was performed based on the features of the LiDAR measurement, with the precision increased via calibration using the IMU/GNSS measurements. Subsequently, the point cloud map was converted into an ND map and used for the real-time localization process. Following the point cloud preprocessing, the pose of the ego vehicle was computed by comparing the ND map and the preprocessed point cloud. In addition, the GPU parallel processing was conducted during the map-matching process. Figure 1 demonstrates the GPU-accelerated NDT localization algorithm flow. The resultant pose of the vehicle was entered into a prediction filter to reduce the computational time by predicting the next pose.

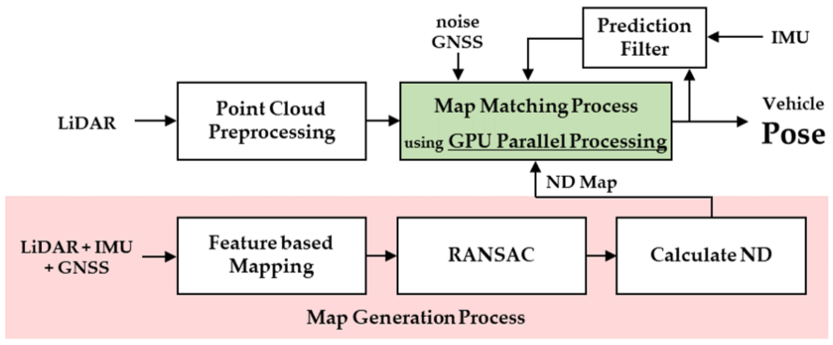

Figure 1. Flowchart of GPU-accelerated NDT localization algorithm.

2.1. Map Generation

2.1.1. Global 3D Point Cloud Map

In this section, we describe the 3D point cloud map generation process. Although the 3D point cloud map generation is not the core focus of this study, it is a preliminary step that is required to create a lightweight map for localization. As the precise 3D point cloud map generation process has been extensively described in various papers, we will only briefly discuss this process.

The process was developed on the basis of the LiDAR-based SLAM algorithm, i.e., LIO-SAM, proposed by Tixiao Shan [34]. However, unlike LOAM, scan matching is conducted using LiDAR features comprising planes and edges [35,36] and IMU tracking. The accumulated errors caused by odometry tracking were eliminated by loop closure, and factor graph searching efficiency was improved using GNSS data. In this study, the 3D point cloud map generation algorithm was produced by modifying the LIO-SAM algorithm according to the research environment. Figure 2 demonstrates the 3D point cloud map of the urban area implemented in the simulation.

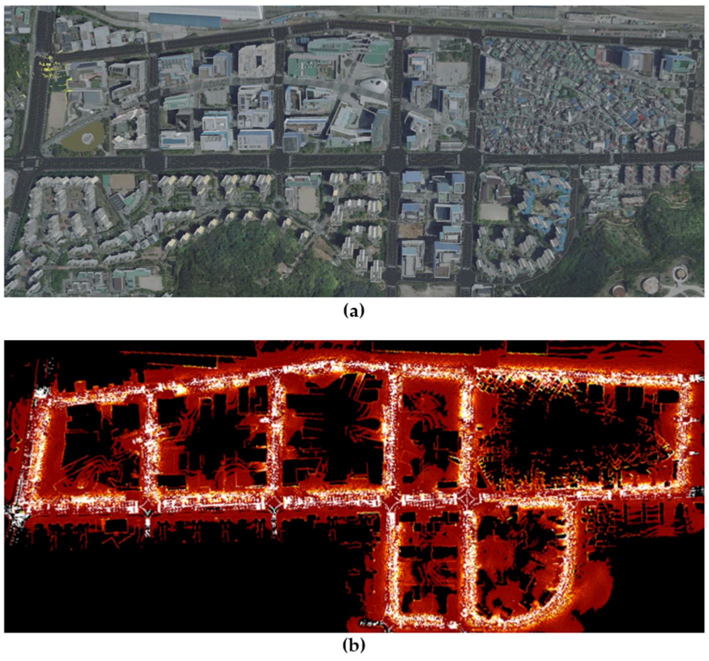

Figure 2. Bird's-eye views of the urban area implemented in the CarMaker simulation. (**a**) CarMaker simulation viewer. (**b**) 3D point cloud map.

To verify the 3D point cloud map generation, lane information from a high-definition road map provided by the National Geographic Information Institute, which had a maximum error of 0.25 m, was utilized [37]. The LiDAR sensor returned 3D coordinates (x, y, z) and reflection (intensity) data; lane information was extracted from the intensity measurements. The coordinate system of the point cloud map was converted into UTM coordinates to allow for a comparison between the common lane information in the two maps. Figure 3 demonstrates the comparison of the intensity measurements of the point cloud map and the lane information from the high-definition road map.

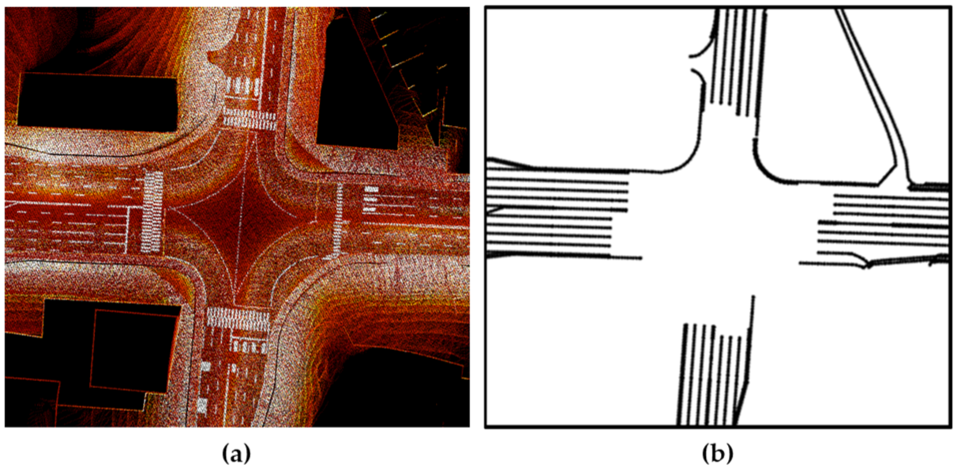

Figure 3. Comparison of road lane information. (**a**) 3D point cloud map. (**b**) High-definition road map.

2.1.2. Vector-Based Normal Distribution Transform

The generated 3D point cloud map extensively described the 3D information of the urban area. Generally, an autonomous driving algorithm requires a three-degree-of-freedom vehicle pose comprising x, y, and yaw, which can be obtained using a 2D map. Therefore, in this study, a 2D ND map was created from the 3D point cloud map. The location and shapes of buildings in an urban area are constant, and the location of autonomous vehicles can be uniquely identified by utilizing the shapes of these buildings using LiDAR measurements. However, some buildings in urban areas have complex 3D shapes. Therefore, multiple layers should be included in the map to represent the 3D shape of buildings, with each layer representing the shape of the cross-sections projected in the 2D plane, thereby enabling multiple layers to capture the 3D shape of a building in a 2D map. Figure 4 shows the vector-based NDT of three representative layers of a set of urban buildings.

After creating the 2D point cloud map, the vectors corresponding to the walls of the buildings were extracted using the random sampling consensus (RANSAC) process. During the RANSAC process, the average vertical distance between each vector and the proximate points of the point cloud were represented using an uncertainty parameter (σ), which can be calculated as follows:

$$\sigma = \frac{1}{n}\sum_{k=1}^{n} \text{Distance}\left(\vec{v}, P_k\right) \tag{1}$$

where $\vec{v}$ is a vector obtained using the RANSAC process and P_k represents summed over positions of the point cloud near the vector. If the vector component was not properly extracted with RANSAC because of noise in the point cloud, the vector was directly designated. The vector and uncertainty information were used to calculate the covariance matrix, which represents the point cloud through NDT, using Equation (2) as follows:

$$\Sigma = U\Lambda U^T = \begin{bmatrix} \lambda_1 v_x^2 + \lambda_2 n_x^2 & \lambda_1 v_x v_y + \lambda_2 n_x n_y \\ \lambda_1 v_x v_y + \lambda_2 n_x n_y & \lambda_1 v_y^2 + \lambda_2 n_y^2 \end{bmatrix} \tag{2}$$

where U is a matrix consisting of eigenvectors defined by $\vec{v}$ and its perpendicular vector, $\vec{n}$, as follows:

$$U = \begin{bmatrix} \vec{v} & \vec{n} \end{bmatrix} = \begin{bmatrix} v_x & n_x \\ v_y & n_y \end{bmatrix} \tag{3}$$

where Λ is a matrix comprising two eigenvalues, i.e., λ_1 and λ_2. Eigenvalue λ_1 was set as half the length of $\vec{v}$, and λ_2 was set as σ.

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} L/2 & 0 \\ 0 & \sigma \end{bmatrix} \tag{4}$$

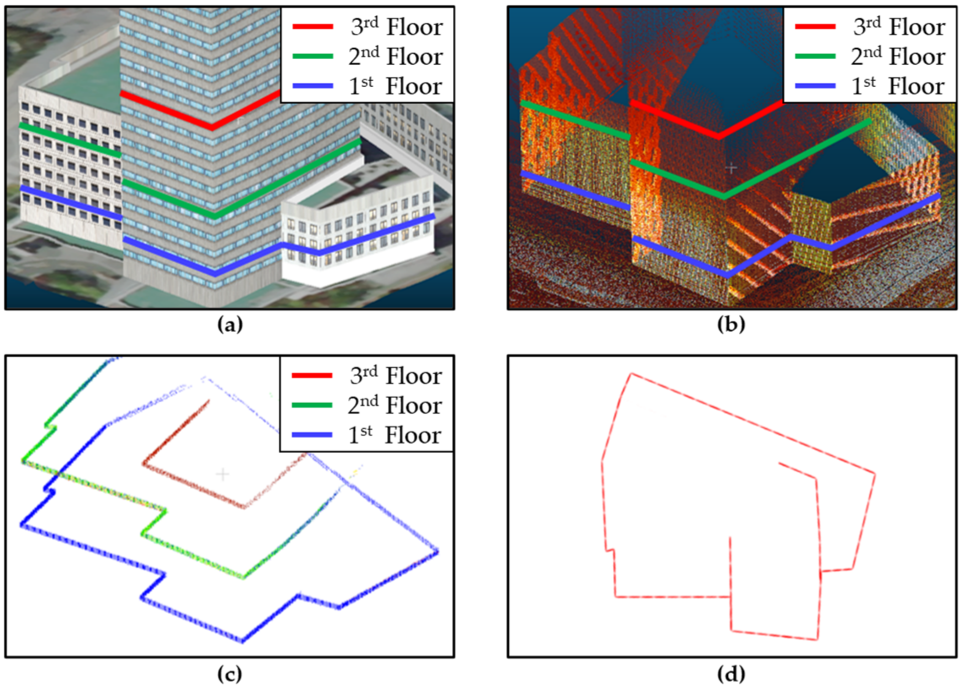

Figure 4. Vector-based NDT process of representing urban buildings. (**a**–**c**) Representative building layers. (**d**) ND map of the building.

Figure 5 demonstrates elements of the ND map used in Equations (1)–(4). The blue arrow in Figure 5a represents the vector that was extracted using RANSAC, and the black points on both sides of the arrow represent the point cloud near the vector. Figure 5b shows an approximation of the point cloud, as measured from the building wall by the ND map. The covariance matrix, i.e., Σ, and the center point of $\vec{v}$, V_c were stored together in the vector-based ND map file. The 3D point cloud map with an original size of 2 GB was reduced to an ND map with a size of 2 MB using the vector-based NDT process.

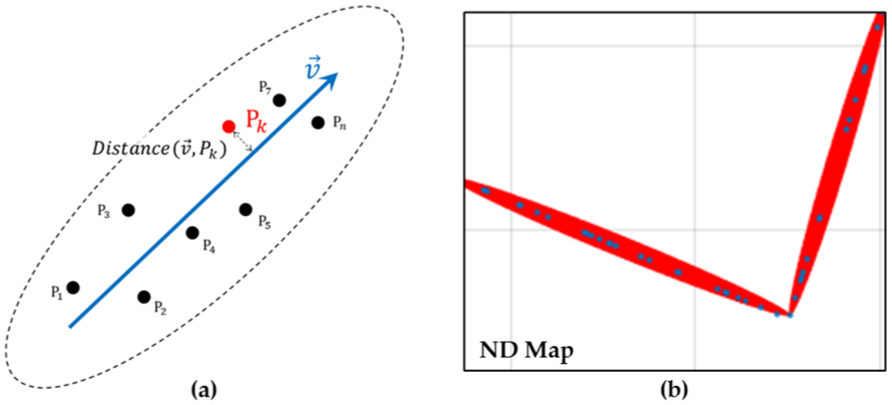

Figure 5. (**a**) Elements used for the generation of the vector-based ND map. (**b**) Part of vector-based ND map and point cloud map

2.2. Map Matching Using GPU Parallel Processing

This study developed a real-time localization algorithm by comparing an ND map with LiDAR measurements. To do so, the LiDAR measurements required several preprocessing steps. First, the points in the cloud up to a height of 4 m from the road's surface were removed to eliminate unnecessary measurements. This preprocessing method rapidly removed measurements originating from the ground and dynamic objects. Subsequently, a 3D point cloud containing only the building information was projected onto a 2D x–y plane. In addition, the real-time LiDAR measurements were not downsampled, thus enhancing the accuracy of the localization result.

The preprocessed 2D point cloud was transferred to the UTM coordinate system using the vehicle pose (position and heading angle). The 2D homogeneous matrix that was used for the transformation of the coordinate is shown in Equation (5).

$$Pose = \begin{bmatrix} x_t \\ y_t \\ \theta_r \end{bmatrix}, \quad M_t = \begin{bmatrix} \cos\theta_r & -\sin\theta_r & x_t \\ \sin\theta_r & \cos\theta_r & y_t \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The transformed point cloud, i.e., $M_t P_k$, was used to determine the nearest vector, v_j, in the ND map. Subsequently, the score was calculated using Equation (6), where C_j represents the center point of v_j and Σ_j represents the covariance matrix of v_j.

$$\text{Score}(M_t, P_k, j) = \exp\left(-\frac{(M_t P_k - C_j)^T \Sigma_j^{-1} (M_t P_k - C_j)}{2}\right) \tag{6}$$

After the score was calculated, the Cost for each M_t was calculated by summing over the scores, as shown in Equation (7). The negative sign was applied to modify the sum of scores as a minimization problem.

$$\text{Cost}(M_t) = -\sum_{k=1}^{n} \text{Score}\ (M_t, P_k, j) \tag{7}$$

Thereafter, Newton's method for optimization was applied to the Cost to repeatedly correct the pose toward the minimum value of the Cost. The calculation of the change in the pose, i.e., $\Delta Pose$, used a Hessian/gradient matrix, as expressed in Equation (8) below.

$$\Delta Pose = \begin{bmatrix} \Delta x_t \\ \Delta y_t \\ \Delta\theta_r \end{bmatrix} = -H^{-1} g, \quad H = \begin{bmatrix} \frac{\partial^2 C}{\partial x^2} & \frac{\partial^2 C}{\partial x \partial y} & \frac{\partial^2 C}{\partial x \partial \theta} \\ \frac{\partial^2 C}{\partial y \partial x} & \frac{\partial^2 C}{\partial y^2} & \frac{\partial^2 C}{\partial y \partial \theta} \\ \frac{\partial^2 C}{\partial \theta \partial x} & \frac{\partial^2 C}{\partial \theta \partial y} & \frac{\partial^2 C}{\partial \theta^2} \end{bmatrix}, \quad g = \begin{bmatrix} \frac{\partial C}{\partial x} \\ \frac{\partial C}{\partial y} \\ \frac{\partial C}{\partial \theta} \end{bmatrix} \tag{8}$$

Figure 6 demonstrates a flowchart of the map-matching process between the ND map and the point cloud. The map-matching process was performed for all points in the cloud and was repeated until an appropriate pose was obtained. The computation time of the overall map-matching process increased with an increase in the number of point clouds. To reduce the overall computation time, this study used GPU parallel processing for the map-matching process.

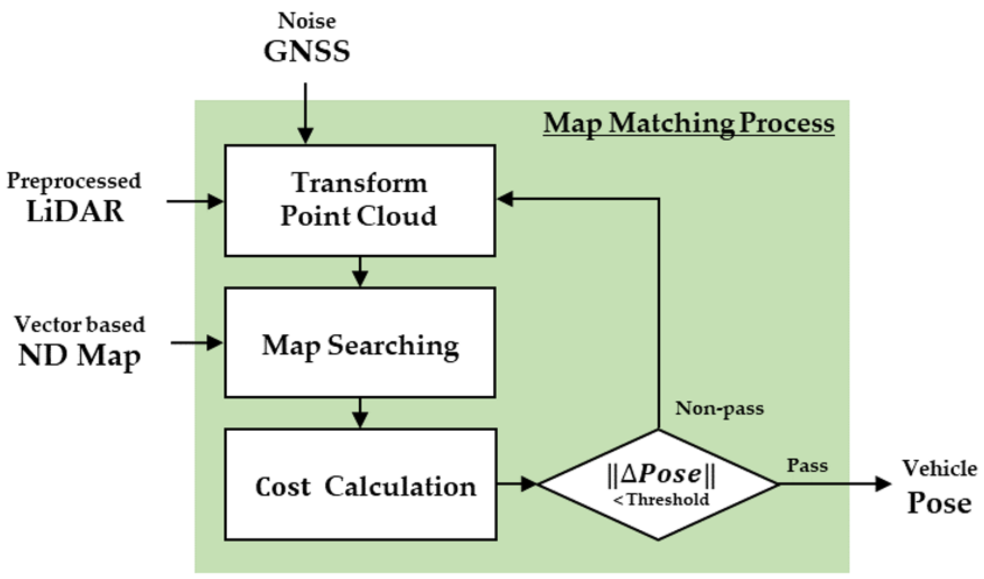

Figure 6. Flowchart of the map-matching process.

For GPU parallel processing, a large amount of the point cloud was moved from the central processing unit (CPU; Host) memory to the GPU (Device) memory, after which the operations on each point were performed parallelly in the GPU thread. This ensured that an operation that needed to be repeated N times on the CPU could be executed at once on the N threads of the GPU. As the size of data to be computed increased, the efficiency of the GPU parallel processing increased compared with that of the CPU serial processing [38]. However, not all operations were suitable for parallel processing. Therefore, to take advantage of parallel processing, the algorithm should comprise numerous simple iterative operations rather than a small number of complex branching operations. The localization algorithm proposed in this study was suitable for GPU parallel processing as the same operations, Equations (5)–(8), were repeated for all points P_k in the point cloud. However, when a large number of individually calculated values from all point clouds had to be added to obtain a single value, bottlenecks were observed in the map-matching process.

2.2.1. GPU Parallel Reduction

In this study, the bottlenecks of the map-matching process were resolved using parallel reduction [39]. Figure 7 shows the computational flow of a cost calculation in which the parallel reduction was applied. To utilize CPU serial processing for the cost calculation, the addition of the score value to the cost value was repeated consecutively. In contrast, the parallel reduction performed a tree-type addition to obtain the final overall sum by repeating the individual additions of two pieces of data (Figure 7). The parallel reduction process reduced the number of operations conducted from N to $\log_2 N$. Algorithm 1 represents the pseudocode of the parallel cost calculation. Different GPU threads were used to simultaneously compute the score between each point and the nearest vector. Next, threads that were separated by a specific index were added together. These computations were conducted for only half of the total threads, and the number of threads subjected to these computations was continuously reduced by half to ensure that the operation was conducted only on the first thread. Following the parallel reduction, the sum of all the scores was computed.

When the parallel reduction was performed, the score data were repeatedly loaded. However, storage of the score data in the global memory resulted in unnecessary repetition of the data transfer delay between the thread and the global memory during operation [40]. Therefore, the score data were transferred from the global memory to the shared memory in advance to minimize the delay caused by data loading.

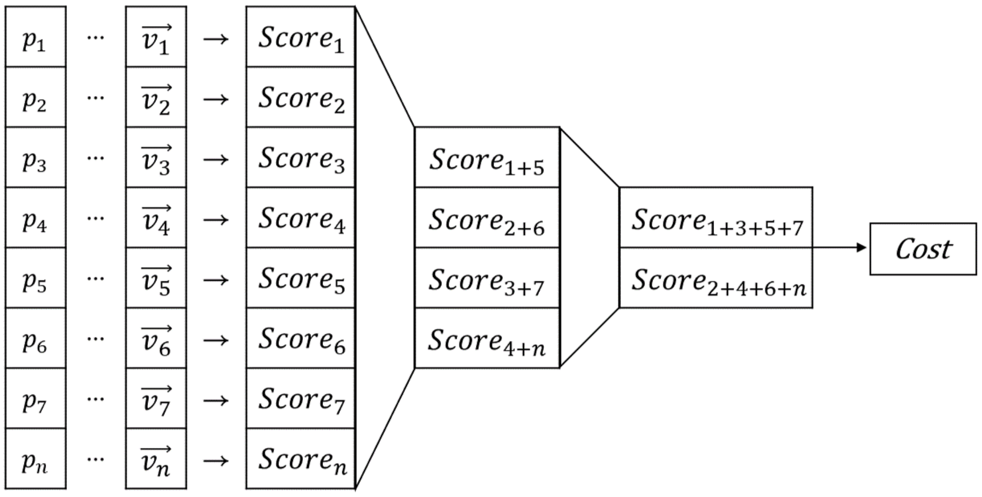
p_1 … $\vec{v_1}$ → $Score_1$
p_2 … $\vec{v_2}$ → $Score_2$
p_3 … $\vec{v_3}$ → $Score_3$
p_4 … $\vec{v_4}$ → $Score_4$
p_5 … $\vec{v_5}$ → $Score_5$
p_6 … $\vec{v_6}$ → $Score_6$
p_7 … $\vec{v_7}$ → $Score_7$
p_n … $\vec{v_n}$ → $Score_n$
$Score_{1+5}$
$Score_{2+6}$
$Score_{3+7}$
$Score_{4+n}$
$Score_{1+3+5+7}$
$Score_{2+4+6+n}$
$Cost$

Figure 7. Operational flow of the parallel cost calculation.

Algorithm 1: Parallel Cost Calculation

```
Input   : Preprocessed points $p$, closest vector $\vec{v}_p$ to each point
Output: $Cost$ calculated from all $p$ and $\vec{v}_p$
begin
    Allocate device memory space for $p$ and $\vec{v}_p$, $Cost$;
    Copy $p$ and $\vec{v}_p$ from host to device global memory;
    do in parallel
        Put every $p$ and $\vec{v}_p$ from global memory to each thread, then compute
          $Score$ between $p$ and $\vec{v}_p$ based on (6);
    end
    do in parallel
        Put $Score$ from global memory to shared memory and get thread index $T_i$;
        for step=length($Score$)/2; step<1; step=step/2 do
            if $T_i$<step then
                $Score[T_i]$+=$Score[T_i$+step];
            end
            Synchronize all threads;
        end
    end
    Copy $Score$ from device global memory to host;
    Extract $Cost$ from first $Score$ and return $Cost$;
end
```

Before conducting the cost calculation, map searching was conducted to determine the closest vector to each point. A parallel reduction was applied to the comparison operation during this process. In contrast to the cost calculation, a parallel reduction was conducted separately for each point. As shown in Figure 8a, different points are loaded in each GPU block, and the distances from these points to all vectors are calculated. Subsequently, parallel reduction was individually conducted for each block, as shown in Figure 8b.

Thereafter, the nearest vector for each block was calculated using the map-searching results. Algorithm 2 represents the pseudocode of the parallel map searching.

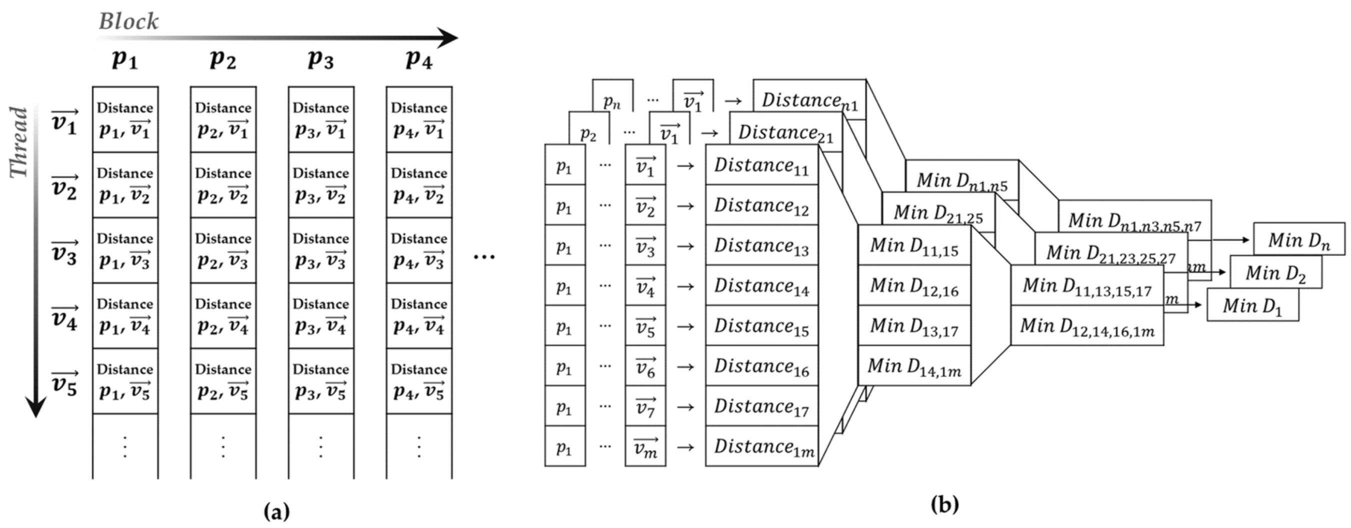

Figure 8. (**a**) Depiction of using GPU thread and block. (**b**) Operational flow of the parallel map searching.

Algorithm 2: Parallel Map Searching

```
Input   : Preprocessed points $p$, vector $\vec{v}$ of ND map
Output: Closest vector $\vec{v}_p$ to each point in $p$
begin
    Allocate device memory space for $p$ and $\vec{v}$, $\vec{v}_p$;
    Copy $p$ and $\vec{v}$ from host to device global memory;
    do in parallel
        Put $p$ from global memory to each block;
        Put $\vec{v}$ from global memory to each thread of block;
        Compute $Distance$ between $p$ and center point of $\vec{v}$ at every thread;
    end
    do in parallel
        Allocate shared memory space for $t\_Distance$ in every block;
        $B_i$=Block index;
        $L_i$=Local thread index in every block;
        $G_i$=Global thread index;
        $t\_Distance[L_i]$=$Distance[G_i]$;
        for step=length($t\_Distance$)/2; step<1; step=step/2 do
            if $L_i$<step then
                $t\_Distance[L_i]$=Min value between $t\_Distance[L_i]$ and
                  $t\_Distance[L_i$+step] ;
            end
            Synchronize all threads;
        end
        At first thread in every block, $\vec{v}_p[B_i]$=$\vec{v}$ of $t\_Distance[0]$;
    end
    Copy $\vec{v}_p$ from device global memory to host and return $\vec{v}_p$;
end
```

3. Experiments

In this study, the proposed algorithm was developed on the basis of a C/C++ robot operating system in Ubuntu 18.04 (Canonical Ltd., London, UK). The computation was accelerated using the GPU parallel processing abilities of the CUDA platform (NVIDIA

Corporation, Santa Clara, CA, USA). Table 1 shows the detailed specifications of the computing system used for verification.

Table 1. Specifications of the computing system used for conducting the verification.

	Laptop	**NVIDIA Jetson Xavier AGX**
CPU	Intel Core i7-9750H	8-core ARM v8.2 64-bit CPU
RAM	16 GB	32 GB
GPU	NVIDIA GeForce RTX2070	512-core Volta GPU
OS	Ubuntu 18.04 + ROS Melodic	
CUDA	Ver.11.2	

The proposed vector-based normal distribution, i.e., VNDT, was compared with other localization methods—specifically, iterative closest point, i.e., ICP, and grid-based normal distribution transform, i.e., GNDT, [32,41]. In this study, we used corresponding algorithms that were implemented on the Point Cloud Library [42]. The tuning parameters common to all algorithms were the maximum number of iterations and the iteration tolerance. The maximum number of iterations was set to 50, and the iteration tolerances were set to 0.1 cm and 0.01°. Detailed descriptions of the tuning parameters for these algorithms are available in [43,44]. In addition, the grid size and vector length are important tuning parameters for NDT algorithms. In this study, these parameters were set to 2 m.

3.1. Open Datasets

To verify the performance of the algorithm with real sensor data, including noise, the nuScenes dataset was used [45]. The nuScenes dataset is a public dataset for autonomous driving developed by an autonomous vehicle company, Motional. Scenes containing complex driving environments in Boston and Singapore were selected from the dataset. The reasons for choosing each scene are as follows.

- scene-0061: The scene data were recorded from Singapore's One North. The driving route at an intersection was under construction. Using the scene data, the accuracy of localization in a congested situation (e.g., passersby standing before traffic signals, construction machine on the road) was verified.
- scene-0103: The scene data were recorded from Boston Seaport. The driving route started at the Congress Street Bridge with no buildings nearby. Using the scene data, the initialization of localization was verified with distant building information.
- scene-1094: The scene data were recorded from Singapore's Queenstown. Since the vehicle was driven on a rainy road at night, the LiDAR measurement was unstable. Using the scene data, the accuracy of localization was verified using the unstable LiDAR measurement.

Each scene included ego pose data that had been precisely calibrated using a point cloud map provided by Motional. In this study, the ND map was produced by stacking point cloud data according to the corresponding ego pose. The driving time for acquiring each scene's data was 20 s, and the data from each sensor (LiDAR, GPS, IMU, etc.) and the calibrated ego pose were stored at a constant rate of 2 Hz. The driving distances for each scene were 91 m (scene-0061), 118 m (scene-0103), and 126 m (scene-1094). Figure 9 demonstrates the driving route of each scene.

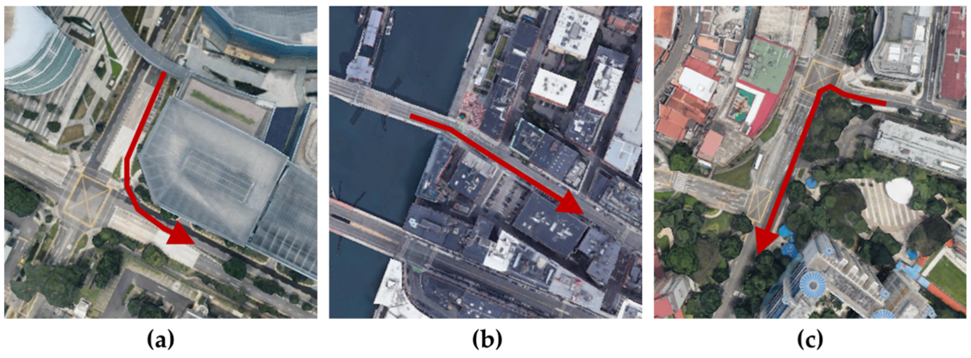

Figure 9. Driving routes (i.e., red arrow) of the selected nuScenes scenes. (**a**) scene-0061, (**b**) scene-0103, and (**c**) scene-1094.

3.2. Simulation

The CarMaker simulation program (IPG Automotive GmbH) was used to verify the computational efficiency with long-term driving. To conduct the localization at the same coordinates as the real urban area, the CarMaker simulation was developed according to the high-definition road map information provided by the National Geographic Information Institute of Korea. It was also possible to implement the simulated buildings with a shape and density very similar to reality by referring to photos of urban areas. Figure 10 shows a comparison of the real and simulated urban areas.

Simulated driving was conducted using a simulation vehicle equipped with LiDAR, GNSS, and IMU sensors in CarMaker. To enhance the realism of the simulation, the vehicle was developed according to the specifications of the KIA Niro. Table 2 lists the detailed specifications.

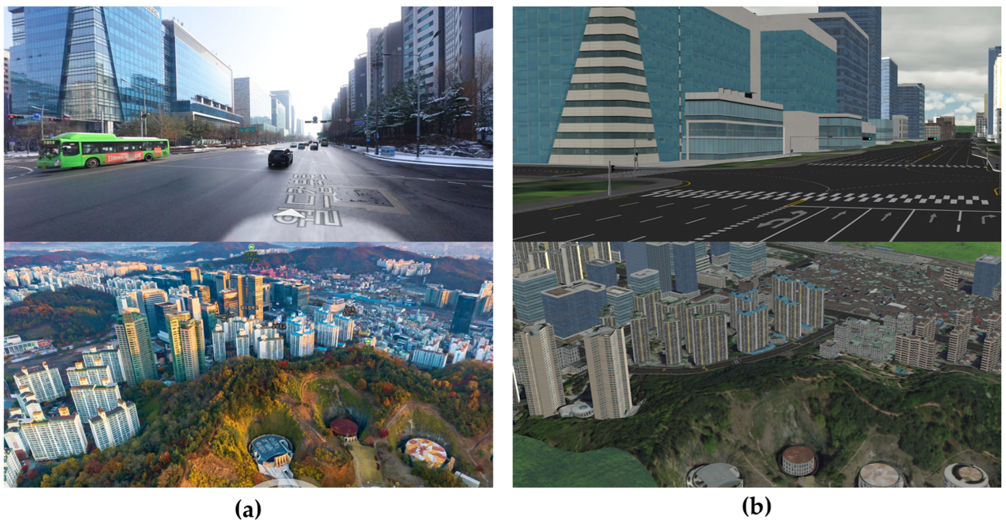

Figure 10. Comparison of the urban area implementation in CarMaker. (**a**) Real urban view. (**b**) CarMaker simulation viewer.

Table 2. Specified parameters of the simulation vehicle.

Designation	Parameter	Designation	Parameter
Overall Length	4735 mm	Wheelbase	2700 mm
Overall Width	1805 mm	Front Tread	1562 mm
Overall Height	1570 mm	Rear Tread	1572 mm
		Curb Weight	1755 kg

Table 3 lists the detailed specifications of the simulation LiDAR sensor, which was implemented by referring to Ouster's OS1-64 model.

Table 3. Specified parameters of the simulation LiDAR.

Designation	Parameter	Designation	Parameter
Vertical Resolution	64 channels	Scan Range	100 m
Horizontal Resolution	1024	Rotation Rate	10 Hz
Vertical Angular Resolution	0.35°	Points per Second	655,360
Vertical Field of View	−22.5°∼22.5°	Data Field	X, Y, Z, Intensity

Considering that there was no error in the GNSS within the simulation, a radial uniform distribution error of 2 m was introduced. The noise in the GNSS reading was used as the initial pose in the localization algorithm, and the original GNSS was used only to represent the ground truth for evaluating the performance of the algorithm. Figure 11 shows the position of the sensors attached to the simulation vehicle. The LiDAR sensor was placed at the center of the front seat of the vehicle and was installed 15 cm above the vehicle roof to reduce the shadow area. The GNSS sensor was attached to the center of the roof above the rear wheel axle, and the IMU sensor, which was used to track the vehicle odometry, was placed at the vehicle's center of gravity.

The length of the verification route was 4300 m, and the total driving time was 7 min. During the simulation, the vehicle moved at a speed of 60 km/h using the CarMaker auto-driving system. Since localization was conducted on all LiDAR inputs (10 Hz), the root-mean-square error (RMSE) was calculated with more than 4000 resultant data points. Figure 12 shows the verification route in CarMaker.

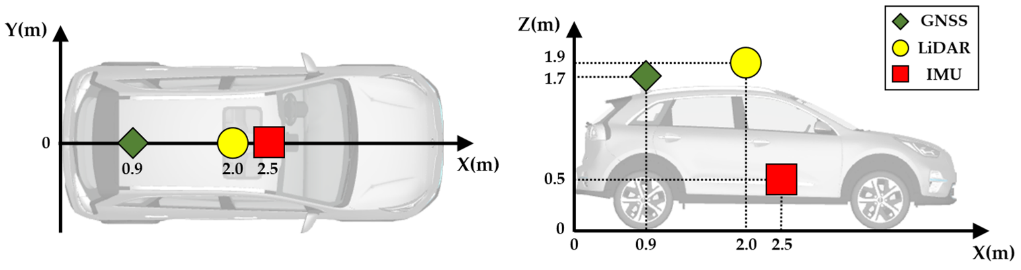

Figure 11. Position of the sensors attached to the simulation vehicle.

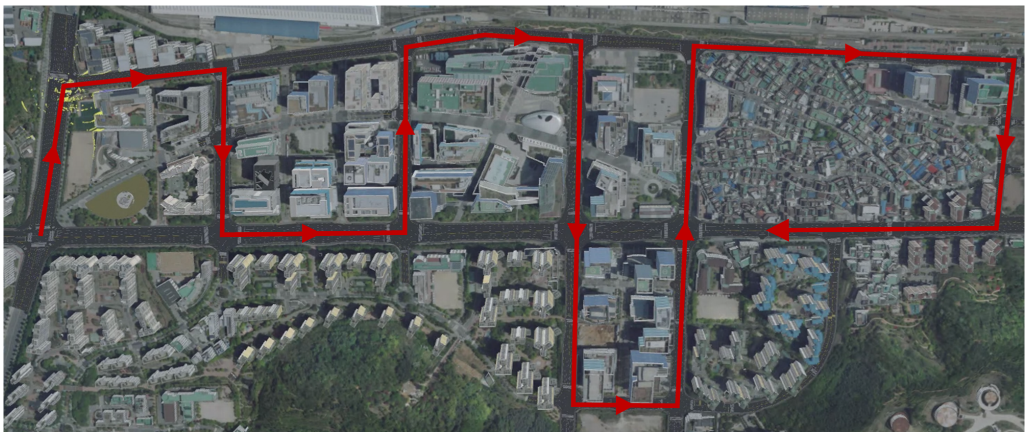

Figure 12. Verification route (i.e., red arrow) of CarMaker.

4. Results and Discussion

4.1. Open Datasets

The top row of Figure 13 shows the localization results from the nuScenes dataset. Compared with the reference ego pose, VNDT demonstrated the highest precision among the localization algorithms. In contrast, ICP had a constant bias, and GNDT was adversely affected by noise. The bottom row of Figure 13 shows the matching results of the LiDAR measurement and ND map in nuScenes. The red ellipses in the figure represent the ND information, and the black points represent the LiDAR measurements. The bird's-eye perspective indicates that the point cloud had been properly matched to each ellipse. Notably, the point cloud was properly matched to the ND map even when the shapes of the buildings were complex and dense.

The RMSE and the maximum localization error were calculated (Table 4). The RMSE in both the longitudinal and lateral directions was within 2 cm, and the heading was within 0.1°. The longitudinal and lateral errors were within 25 cm; this meant that the performance of the algorithms did not exceed the localization error value recommended by ISO 17572.

Table 5 lists the average computation time for each localization algorithm. The proposed algorithm (VNDT using GPU) achieved a real-time performance suitable for use with 10–20 Hz LiDAR and increased the computational speed of the embedded computer almost 100-fold. The computation time of the GPU parallel processing functions proposed in Section 2.2.1 was analyzed in a CarMaker simulation for verification because the nuScenes dataset included only 30 data points in one scene.

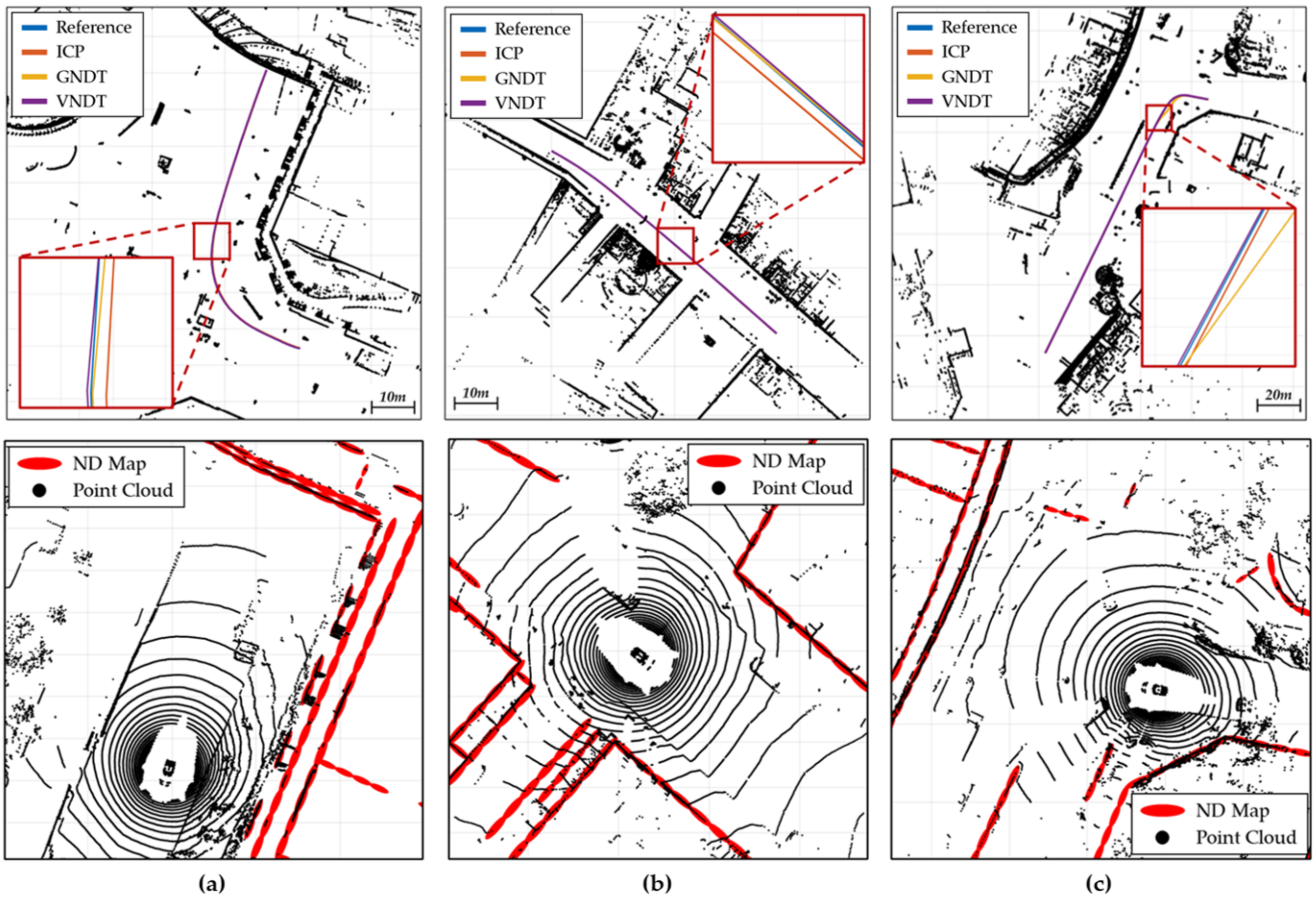

Figure 13. Comparison between the localization results and the reference (**top**) and LiDAR map-matching results (**bottom**) in nuScenes. (**a**) scene-0061, (**b**) scene-0103, and (**c**) scene-1094.

Table 4. Accuracy comparison of the localization methods in nuScenes.

		Localization Error								
		Longitudinal			Lateral			Heading		
		ICP	GNDT	VNDT	ICP	GNDT	VNDT	ICP	GNDT	VNDT
scene-0061	**RMSE**	5.57 cm	2.78 cm	**1.40 cm**	7.17 cm	3.16 cm	**1.81 cm**	0.03°	0.08°	**0.02°**
	MAX. dev.	12.26 cm	9.16 cm	**4.13 cm**	14.92 cm	11.92 cm	**4.31 cm**	0.09°	0.31°	**0.06°**
scene-0103	**RMSE**	3.76 cm	3.27 cm	**1.92 cm**	9.01 cm	2.58 cm	**1.54 cm**	0.02°	0.03°	**0.02°**
	MAX. dev.	6.95 cm	12.92 cm	**4.9 cm**	5.04 cm	10.75 cm	**3.39 cm**	0.10°	0.13°	**0.05°**
scene-1094	**RMSE**	6.05 cm	20.8 cm	**1.59 cm**	4.76 cm	17.23 cm	**1.44 cm**	0.05°	0.10°	**0.02°**
	MAX. dev.	17.44 cm	140.93 cm	**4.58 cm**	11.31 cm	136.31 cm	**6.76 cm**	0.12°	0.82°	**0.08°**

Table 5. Computation time comparison of localization methods in nuScenes.

		ICP	GNDT	VNDT(CPU)	VNDT(GPU)
scene-0061	**Xavier**	401.98 ms	457.10 ms	4934.22 ms	**51.82 ms**
	Laptop	134.76 ms	145.79 ms	1263.21 ms	**28.35 ms**
scene-0103	**Xavier**	1036.08 ms	946.25 ms	5723.73 ms	**60.12 ms**
	Laptop	340.51 ms	305.23 ms	1387.25 ms	**34.91 ms**
scene-1094	**Xavier**	726.92 ms	612.30 ms	4573.10 ms	**47.63 ms**
	Laptop	219.88 ms	191.57 ms	1300.39 ms	**28.95 ms**

4.2. Simulation

Figure 14 shows the localization result from the CarMaker simulation. The ICP and GNDT algorithms failed to localize the vehicle in an area where small buildings were densely located. Therefore, it is not possible to analyze the precision and calculation time of the ICP and GNDT algorithms. Table 6 lists the accuracy of the VNDT algorithm.

Table 6. Accuracy of the proposed localization method in CarMaker.

	Localization Error		
	Longitudinal	Lateral	Heading
RMSE	2.21 cm	2.92 cm	0.12°
MAX. dev.	6.84 cm	7.38 cm	0.31°

Table 7 compares the computation time between the CPU and GPU-based VNDT algorithms. Figure 15 shows a comparison of the average computation time of the GPU parallel processing functions (i.e., transform point cloud, map searching, and cost calculation). The computation time of each function includes the length of the data transfer delay between the CPU and GPU memory.

Table 7. Computation time comparison of localization methods in CarMaker.

	VNDT(CPU)	VNDT(GPU)
Xavier	4125.16 ms	**43.81 ms**
Laptop	1271.98 ms	**26.11 ms**

Figure 16 shows the computation time for each function in the driving time. The figure indicates that the computation time using CPU serial processing increased with an increase in the size of the point cloud after preprocessing. In contrast, the amount of data had no remarkable effect on the computational speed of the GPU parallel processing.

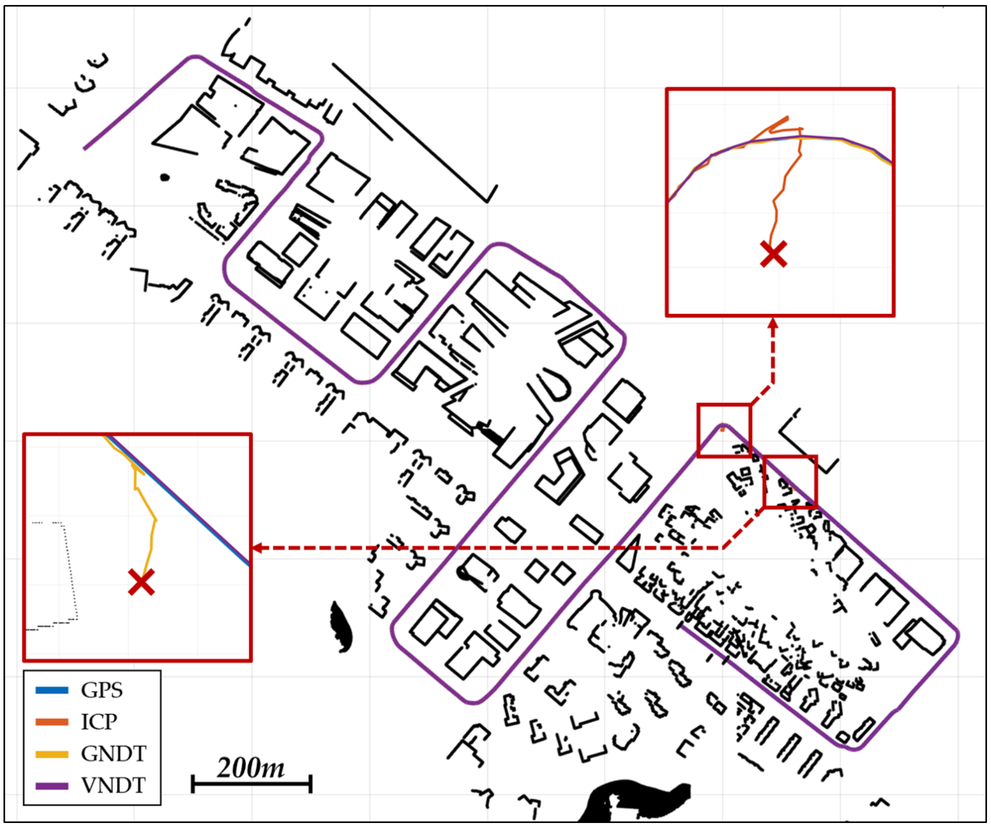

Figure 14. Localization result from the CarMaker simulation.

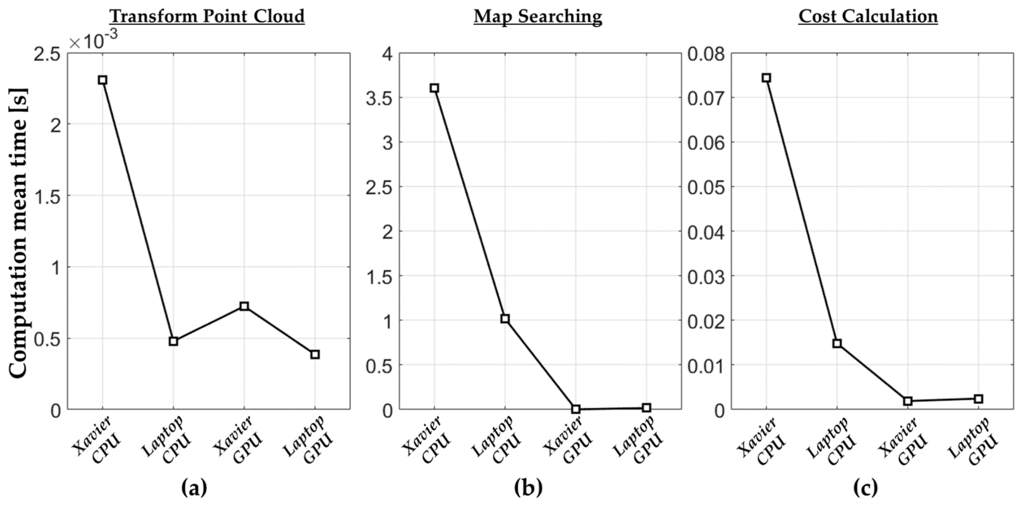

Figure 15. The average computation time for each function in the CarMaker simulation. (**a**) Transform point cloud. (**b**) Map searching. (**c**) Cost calculation.

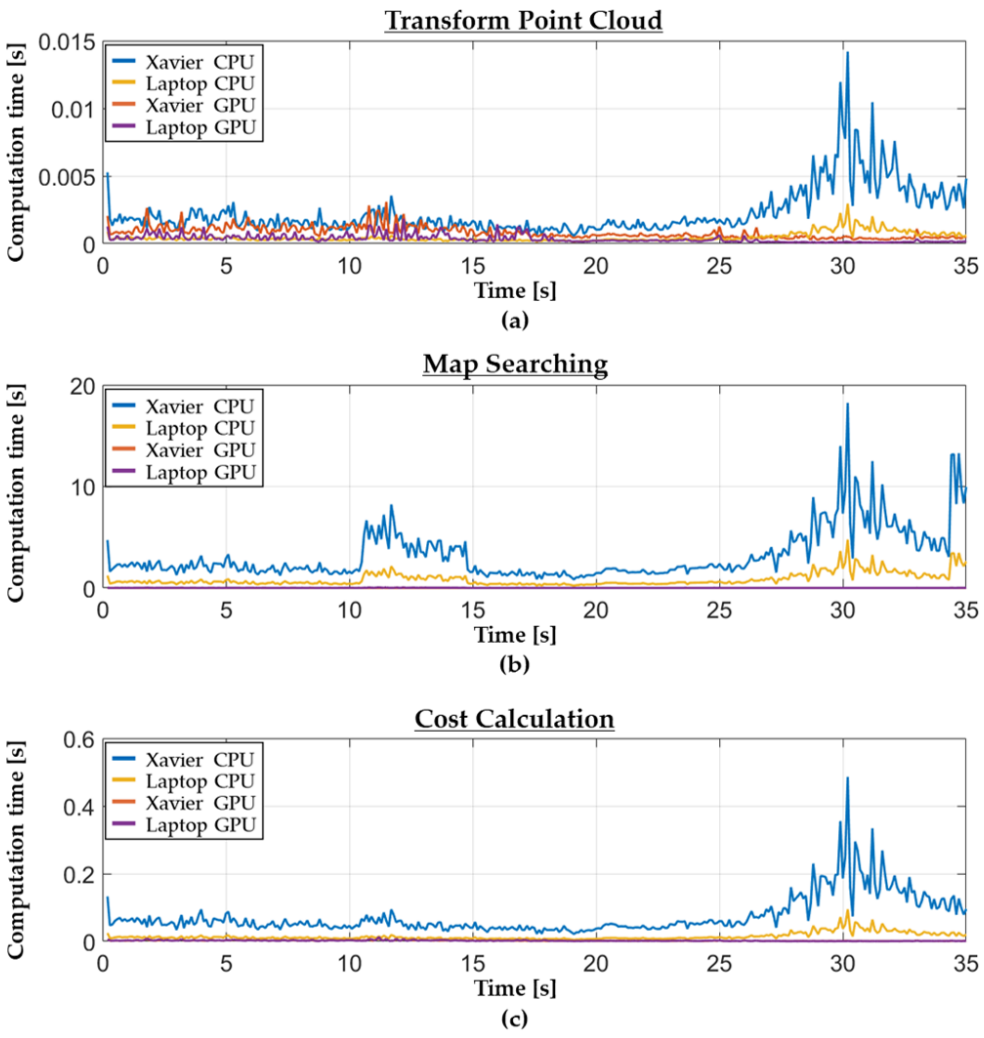

Figure 16. Computation time for each function in the CarMaker simulation. (**a**) Transform point cloud. (**b**) Map searching. (**c**) Cost calculation.

5. Conclusions

In this study, we introduced a GPU-accelerated vector-based NDT localization algorithm that could guarantee precision and real-time performance for 10–20 Hz high-resolution LiDAR, even in GNSS-denied urban areas. Therefore, a vector-based ND map suitable for the parallel processing algorithm was used for the reference map. Parallel processing was applied to the map-matching process to maintain a stable and fast computational speed, regardless of the size of the LiDAR measurement data. In particular, the parallel reduction was used for map searching and the cost calculation function of the map-matching process. The performance of the proposed algorithm was evaluated with the nuScenes open dataset and urban environment in a CarMaker simulation. The performance of the proposed algorithm satisfied the accuracy required by ISO 17572. In addition, the fast computational speed of the algorithm enabled a localization update rate of 10 Hz. Therefore, the proposed algorithm exhibited enhanced computational speed and highly reliable performance. In a future study, we plan to study localization in the actual urban area that was referenced in the simulation. We will propose a robust localization algorithm for the real environment and analyze the GPU efficiency in detail.

Author Contributions: Conceptualization, K.W.J. and W.J.J.; Data curation, K.W.J.; Formal analysis, K.W.J.; Funding acquisition, Y.K.; Investigation, K.W.J. and Y.K.; Methodology, K.W.J.; Project administration, Y.K.; Resources, Y.K.; Software, K.W.J.; Validation, K.W.J.; Visualization, K.W.J.; Writing—original draft, K.W.J.; Writing—review & editing, K.W.J. and Y.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Basic Science Research Program through the National Research Foundation of Korea, funded by the Ministry of Education, Science and Technology (2021R1A2C2003254). This research was supported by the Basic Science Research Program through the National Research Foundation of Korea, funded by the Ministry of Education (5199990814084), and the Competency Development Program for Industry Specialists of Korean Ministry of Trade, Industry and Energy, operated by Korea Institute for Advancement of Technology (No. N0002428, HRD program for Future Car), partly.

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

References

1. Badue, C.; Guidolini, R.; Carneiro, R.V.; Azevedo, P.; Cardoso, V.B.; Forechi, A.; Jesus, L.; Berriel, R.; Paixao, T.M.; Mutz, F.; et al. Self-driving cars: A survey. *Expert Syst. Appl.* **2021**, *165*, 113816. [CrossRef]
2. Bimbraw, K. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In Proceedings of the 2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Colmar, France, 21–23 July 2015; Volume 1, pp. 191–198.
3. Schoettle, B.; Sivak, M. *A Survey of Public Opinion about Autonomous and Self-Driving Vehicles in the US, the UK, and Australia*; Technical Report; University of Michigan, Transportation Research Institute: Ann Arbor, MI, USA, 2014.
4. Lee, C.; Ward, C.; Raue, M.; D'Ambrosio, L.; Coughlin, J.F. Age differences in acceptance of self-driving cars: A survey of perceptions and attitudes. In *International Conference on Human Aspects of IT for the Aged Population*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 3–13.
5. Liu, P.; Zhang, Y.; He, Z. The effect of population age on the acceptable safety of self-driving vehicles. *Reliab. Eng. Syst. Saf.* **2019**, *185*, 341–347. [CrossRef]
6. Kocić, J.; Jovičić, N.; Drndarević, V. Sensors and sensor fusion in autonomous vehicles. In Proceedings of the 2018 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 20–21 November 2018; pp. 420–425.
7. Hofmann-Wellenhof, B.; Lichtenegger, H.; Wasle, E. *GNSS–Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and More*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2007.
8. Blomenhofer, H.; Ehret, W.; Leonard, A.; Blomenhofer, E. GNSS/Galileo global and regional integrity performance analysis. In Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2004), Long Beach, CA, USA, 21–24 September 2004; pp. 2158–2168.
9. Kaplan, E.D.; Hegarty, C. *Understanding GPS/GNSS: Principles and Applications*; Artech House: Washington, DC, USA, 2017.
10. Yang, Y.; Mao, Y.; Sun, B. Basic performance and future developments of BeiDou global navigation satellite system. *Satell. Navig.* **2020**, *1*, 1. [CrossRef]
11. ISO 17572-1:2015—Intelligent Transport Systems (ITS)—Location Referencing for Geographic Databases—Part 1: General Requirements and Conceptual Model. Available online: https://www.iso.org/standard/63400.html (accessed on 7 February 2022).
12. ISO 17572-2:2018—Intelligent Transport Systems (ITS)—Location Referencing for Geographic Databases—Part 2: Pre-Coded Location References (Pre-coded Profile). Available online: https://www.iso.org/standard/69468.html (accessed on 7 February 2022).
13. Groves, P.D. Shadow matching: A new GNSS positioning technique for urban canyons. *J. Navig.* **2011**, *64*, 417–430. [CrossRef]
14. Kok, M.; Hol, J.D.; Schön, T.B. Using inertial sensors for position and orientation estimation. *arXiv* **2017**, arXiv:1704.06053.
15. Lee, N.; Ahn, S.; Han, D. AMID: Accurate magnetic indoor localization using deep learning. *Sensors* **2018**, *18*, 1598. [CrossRef] [PubMed]
16. Jiménez, A.; Seco, F. *Ultrasonic Localization Methods for Accurate Positioning*; Instituto de Automatica Industrial: Madrid, Spain, 2005.
17. Fu, Q.; Yu, H.; Wang, X.; Yang, Z.; Zhang, H.; Mian, A. FastORB-SLAM: A fast ORB-SLAM method with Coarse-to-Fine descriptor independent keypoint matching. *arXiv* **2020**, arXiv:2008.09870.
18. Campos, C.; Elvira, R.; Rodríguez, J.J.G.; Montiel, J.M.; Tardós, J.D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Trans. Robot.* **2021**, *37*, 1874–1890. [CrossRef]
19. Mur-Artal, R.; Montiel, J.M.M.; Tardos, J.D. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Trans. Robot.* **2015**, *31*, 1147–1163. [CrossRef]
20. Davison, A.J.; Reid, I.D.; Molton, N.D.; Stasse, O. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *29*, 1052–1067. [CrossRef] [PubMed]

21. Hess, W.; Kohler, D.; Rapp, H.; Andor, D. Real-time loop closure in 2D LIDAR SLAM. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 1271–1278.
22. Zhang, J.; Singh, S. LOAM: Lidar Odometry and Mapping in Real-time. In Proceedings of the Robotics: Science and Systems, Berkeley, CA, USA, 12–16 July 2014; Volume 2, pp. 1–9.
23. Deschaud, J.E. IMLS-SLAM: Scan-to-model matching based on 3D data. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 2480–2485.
24. Ye, H.; Chen, Y.; Liu, M. Tightly coupled 3d lidar inertial odometry and mapping. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 3144–3150.
25. Leitinger, E.; Meyer, F.; Tufvesson, F.; Witrisal, K. Factor graph based simultaneous localization and mapping using multipath channel information. In Proceedings of the 2017 IEEE International Conference on Communications Workshops (ICC Workshops), Paris, France, 21–25 May 2017; pp. 652–658.
26. Dellaert, F. *Factor Graphs and GTSAM: A Hands-On Introduction*; Technical Report; Georgia Institute of Technology: Atlanta, GA, USA, 2012.
27. Whelan, T.; Kaess, M.; Fallon, M.; Johannsson, H.; Leonard, J.; McDonald, J. Kintinuous: Spatially Extended Kinectfusion. In Proceedings of the RSS'12 Workshop on RGB-D: Advanced Reasoning with Depth Cameras, Berkeley, CA, USA, 19 July 2012. Available online: https://dspace.mit.edu/handle/1721.1/71756 (accessed on 7 January 2022).
28. Wolcott, R.W.; Eustice, R.M. Fast LIDAR localization using multiresolution Gaussian mixture maps. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 2814–2821.
29. Wang, L.; Zhang, Y.; Wang, J. Map-based localization method for autonomous vehicles using 3D-LIDAR. *IFAC-PapersOnLine* **2017**, *50*, 276–281. [CrossRef]
30. Yoneda, K.; Tehrani, H.; Ogawa, T.; Hukuyama, N.; Mita, S. Lidar scan feature for localization with highly precise 3-D map. In Proceedings of the 2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, 8–11 June 2014; pp. 1345–1350.
31. Cao, V.H.; Chu, K.; Le-Khac, N.A.; Kechadi, M.T.; Laefer, D.; Truong-Hong, L. Toward a new approach for massive LiDAR data processing. In Proceedings of the 2015 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM), Fuzhou, China, 8–10 July 2015; pp. 135–140.
32. Biber, P.; Straßer, W. The normal distributions transform: A new approach to laser scan matching. In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453), Las Vegas, NV, USA, 27–31 October 2003; Volume 3, pp. 2743–2748.
33. Javanmardi, E.; Javanmardi, M.; Gu, Y.; Kamijo, S. Autonomous vehicle self-localization based on multilayer 2D vector map and multi-channel LiDAR. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 437–442.
34. Shan, T.; Englot, B.; Meyers, D.; Wang, W.; Ratti, C.; Rus, D. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24–30 October 2020; pp. 5135–5142.
35. Im, J.H.; Im, S.H.; Jee, G.I. Vertical corner feature based precise vehicle localization using 3D LIDAR in urban area. *Sensors* **2016**, *16*, 1268. [CrossRef] [PubMed]
36. Zheng, H.; Wang, R.; Xu, S. Recognizing street lighting poles from mobile LiDAR data. *IEEE Trans. Geosci. Remote Sens.* **2016**, *55*, 407–420. [CrossRef]
37. High Definition Road Map of Seoul. Available online: http://map.ngii.go.kr/mn/mainPage.do (accessed on 7 February 2022).
38. Nickolls, J.; Dally, W.J. The GPU computing era. *IEEE Micro* **2010**, *30*, 56–69. [CrossRef]
39. Harris, M. Optimizing parallel reduction in CUDA. *Nvidia Dev. Technol.* **2007**, *2*, 70.
40. Sanders, J.; Kandrot, E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*; Addison-Wesley Professional: Boston, MA, USA, 2010.
41. Magnusson, M. The Three-Dimensional Normal-Distributions Transform: An Efficient Representation for Registration, Surface Analysis, and Loop Detection. Ph.D. Thesis, Örebro Universitet, Örebro, Sweden, 2009.
42. Rusu, R.B.; Cousins, S. 3D is here: Point Cloud Library (PCL). In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011.
43. pcl::IterativeClosestPoint Class Template Reference. Available online: https://pointclouds.org/documentation/classpcl_1_1_iterative_closest_point.html (accessed on 7 February 2022).
44. pcl::NormalDistributionsTransform Class Template Reference. Available online: https://pointclouds.org/documentation/classpcl_1_1_normal_distributions_transform.html (accessed on 7 February 2022).
45. Caesar, H.; Bankiti, V.; Lang, A.H.; Vora, S.; Liong, V.E.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; Beijbom, O. nuScenes: A multimodal dataset for autonomous driving. *arXiv* **2019**, arXiv:1903.11027.