An Edge-Server Partitioning Method for 3D LiDAR SLAM on FPGAs

Mizuki Yasuda*, Keisuke Sugiura*, Ryuto Kojima*, Hiroki Matsutani*
*Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: {yasuda,sugiura,kojima,matutani}@arc.ics.keio.ac.jp

***Abstract*—3D LiDAR SLAM (Simultaneous Localization And Mapping) is utilized for autonomous driving and autonomous mobile robots. When compute resources on an edge device are limited, SLAM is executed by using the edge and server cooperatively. In this paper, we propose an edge-server cooperative 3D LiDAR SLAM based on the state-of-the-art method, LIO-SAM. In the edge-server cooperative SLAM, data transfer between them imposes an overhead. We partition the LIO-SAM method to reduce the data transfer while meeting the time constraints; the edge part transfers a set of feature points instead of a raw LiDAR scan, and the server handles the other parts. Moreover, the edge tasks are accelerated by an FPGA implementation to meet an execution time constraint on the edge side. Specifically, we implement a point cloud deskew on the FPGA. Average execution time on the edge side is accelerated up by 1.63 to 1.73 times. Furthermore, all the LiDAR scans are processed within 100ms which is the execution time constraint on the edge side based on 10Hz LiDAR frequency. We also evaluate the data transfer size and the execution time in detail.**

***Index Terms*—3D LiDAR SLAM, LIO-SAM, Edge computing, FPGA**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamenal technology that is important for mobile robots and autonomous driving cars. SLAM is categorized into two types: Visual SLAM and LiDAR SLAM. Visual SLAM uses RGB camera or RGBD camera as a main external sensor, and uses camera images to estimate the location of a robot. On the other hand, the latter uses LiDAR (Light Detection and Ranging) as a main sensor. LiDAR is a sensor that measures distances between the sensor and objects based on the time-of-flight principle. One advantage of LiDAR sensors is that, an error of distance between a sensor and an object is small and enables high accuracy than Visual SLAM. The LiDAR SLAM systems also have several other advantages. The LiDAR sensors can acquire range data directly and have a 360-degree field of view unlike the cameras. They are unaffected by the lighting due to the use of the lasers. The point clouds acquired from the LiDARs are less blurred, unlike the out of focus images captured from the cameras.

LIO-SAM [1] is a state-of-the-art 3D LiDAR SLAM algorithm based on the famous LOAM [2] algorithm and we use it as a target SLAM algorithm since it achieves high accuracy and better realtime performance compared to the other LOAM-based methods, e.g., LeGO-LOAM [3] and LIOM [4], by incorporating graph optimization and loop closure. LOAM has high accuracy and realtime performance to a certain extent. However, accuracy of LOAM is likely to decrease due to an absence of graph optimization and loop closure, which reduce the cumulative errors and are important for long-term operations.

3D LiDAR SLAM algorithms including LIO-SAM involve point cloud registration which is a function for finding a transformation between two point clouds. Accuracy of the registration significantly affects performance of SLAM. LIO-SAM optimizes the transformation using the Levenberg-Marquardt method. A rich compute resource is needed to cope with the heavy calculation cost and execute this optimization process in realtime. If a rich compute resource is needed on the edge side, the size, price, and power consumption of edge devices become problems. SLAM systems can be executed on edge devices and on servers cooperatively to avoid these problems. Those such as [5] and [6] execute computationally expensive processes such as global map optimization on the server. In this paper, we propose an edge-server cooperative 3D LiDAR SLAM based on LIO-SAM as shown in Figure 1. This approach also has the advantage that multiple SLAM processes and other tasks such as path planning and object recognition can be executed together on the same server.

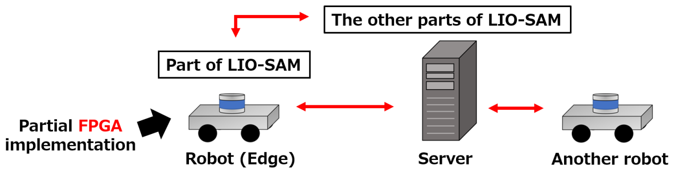

Fig. 1. Edge-server cooperative LIO-SAM

By using FPGAs as an edge device it is possible to improve the memory utilization and execution time. There exist numerous studies that employ FPGAs to accelerate various robotic tasks and improve the overall performance. The point cloud registration is a computationally expensive process, so it is effective to accelerate the registration by FPGA for realtime performance. For example, implementing Correlative Scan Matching (CSM) [7] on an FPGA makes three types of 2D LiDAR SLAM faster without reducing accuracy [8]. Not only 2D LiDAR based but also 3D LiDAR based SLAM is accelerated by FPGAs. In [9], 3D LiDAR based SLAM is accelerated for embedded robotics. They implement the point cloud registration and map update on an FPGA, and accelerate these processes by running in parallel and in a pipelined manner. In [10], K-Nearest-Neighbor search (KNN) for 3D LiDAR SLAM is accelerated for smart vehicles. P3Net [11] is a PointNet-based 2D and 3D path planning for FPGAs, and PointNet [12] is accelerated by the FPGA implementation.

The data transfer becomes a major bottleneck when the raw

LiDAR point clouds are sent to the server without preprocessing, because LiDAR sensor obtains 10^5 to 10^6 points per second and the data size of the point clouds becomes very large and requires a high network bandwidth. It is important to reduce the data transfer by processing the point cloud data in the edge device. In this paper, we partition LIO-SAM nodes [1] to the edge and the server to reduce the data transfer. Another important point on the edge-server cooperative 3D LiDAR SLAM is that the process on the edge devices should meet the realtime constraint to stabilize the SLAM system. In this paper, in addition to partitioning LIO-SAM based on the data transfer size, we accelerate the edge process by offloading the point cloud deskew process onto FPGA.

The rest of this paper is organized as follows. Section II overviews SLAM and LIO-SAM. Section III proposes the edge-server partitioning of LIO-SAM to reduce the data transfer, and accelerates the edge process by FPGA implementation. Section IV describes details of the FPGA implementation. Section V evaluates the proposed approach in terms of the data transfer size and the execution time. Finally, Section VI concludes this paper and mentions future work.

II. Preliminaries

A. *SLAM Overview*

Point cloud registration is one of the most important process in LiDAR SLAM and affects the performance of SLAM. The registration finds a relative pose which consists of relative translation and rotation between two point clouds. There are various point cloud registration algorithms, such as the famous ICP [13] and NDT [14]. ICP is an iterative method which alternates between two processes, associating two point clouds and updating the pose. Point cloud poses are updated to minimize the squared distances between matched points by using optimization methods such as the Newton's method, the Gauss-Newton method, and the Levenberg-Marquardt method.

LiDAR odometry is a process to estimate the robot poses using a sequence of LiDAR scans. There are other types of odometry. For example, IMU odometry calculates the pose from acceleration and angular velocity data from Inertial Measurement Unit (IMU), and wheel odometry calculates the posture from the rotation angles of wheels.

The software implementations of many LiDAR SLAM algorithms, e.g., LOAM [2], are publicly available. These implementations are distributed in the form of ROS packages. ROS is a famous software platform for robots. Our proposed method is also based on ROS. In LOAM, odometry and mapping are implemented in a separate module, i.e., ROS node. Odometry is executed at a high operating frequency and mapping is executed at a low frequency. Realtime performance and accuracy of LOAM are relatively high due to this separated design. However, accuracy of LOAM decreases in a long-term operation because LOAM does not employ graph optimization and loop closures and thus accumulates errors. Various LOAM-based methods have been proposed in the literature to address this issue, such as LeGO-LOAM [3], LIOM [4], and LIO-SAM [1]. LeGO-LOAM is a lightweight and ground optimized SLAM for ground vehicles. In LeGO-LOAM, a pose estimation process is separated into two parts: registration using the ground features and one using the other features. LeGO-LOAM also introduces loop closures and graph optimization. LIOM processes all the sensor data from LiDAR and IMU to achieve high accuracy at the expense of low realtime performance. Compared to that, LIO-SAM achieves both high realtime performance and accuracy. LIO-SAM introduces efficient optimization by introducing a factor graph, which incorporates LiDAR, IMU, and GPS data, and an efficient registration method. Optimization based on the factor graph is formulated as a maximum a posteriori problem.

[1]The ROS node is a process launched in the Robot Operating System (ROS) environment that performs a specific task and communicates with other nodes via messages.

Systems such as CT-ICP [15] and ELO [16] can efficiently estimate accurate LiDAR odometry which use only LiDAR sensors as an external sensor. Such SLAM systems tend to be less accurate than those using multiple sensors, such as LIO-SAM. There exist deep learning based 3D LiDAR SLAM systems, such as LO-Net [17]. It achieves a performance comparable to LOAM in terms of accuracy and realtime performance. However, the deep learning based SLAM generally has a challenge to versatility. It requires a large amount of training data and is difficult to achieve high accuracy in a new location which is not similar to the training data.

Many 3D LiDAR SLAM implementations, such as LOAM [2] and LIO-SAM [1], assume the use of mechanical rotating LiDARs as shown in Figure 2. A rotating LiDAR consists of multiple laser sensors positioned at equally spaced elevation angles, and obtains a scan data composed of all direction points by rotating these sensors. Points obtained from a single sensor is called ring and they have the same elevation angle. Let θ and R denote the resolution of azimuth angles and the number of rings, respectively. The maximum number of the points in one scan is $\frac{2\pi R}{\theta}$. In practice, the number of the points in one scan is smaller than this number, because laser reflections are not detected when there is no object or there is a specific material object in laser direction. There exist other types of LiDAR sensors, such as solid-state LiDARs. LiLi-OM [18], which can utilize a solid-state LiDAR as a main sensor, is a state-of-the-art 3D LiDAR inertial SLAM algorithm in terms of accuracy and execution time.

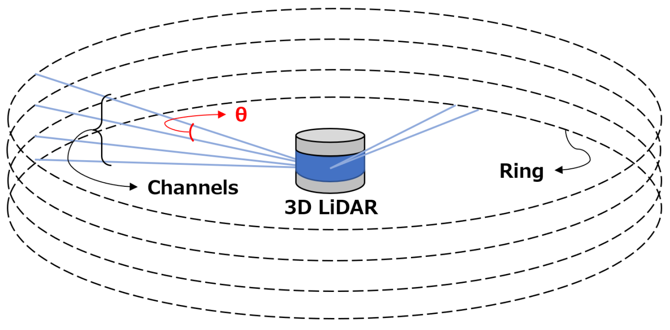

Fig. 2. The 3D LiDAR

Scans obtained from rotating LiDARs often need to be deskewed using the data acquired from the other sensors such as IMU, since the robot moves when acquiring scans. This point cloud deskewing is a process of correcting distortion of the point cloud. If data is not deskewed, accuracy of

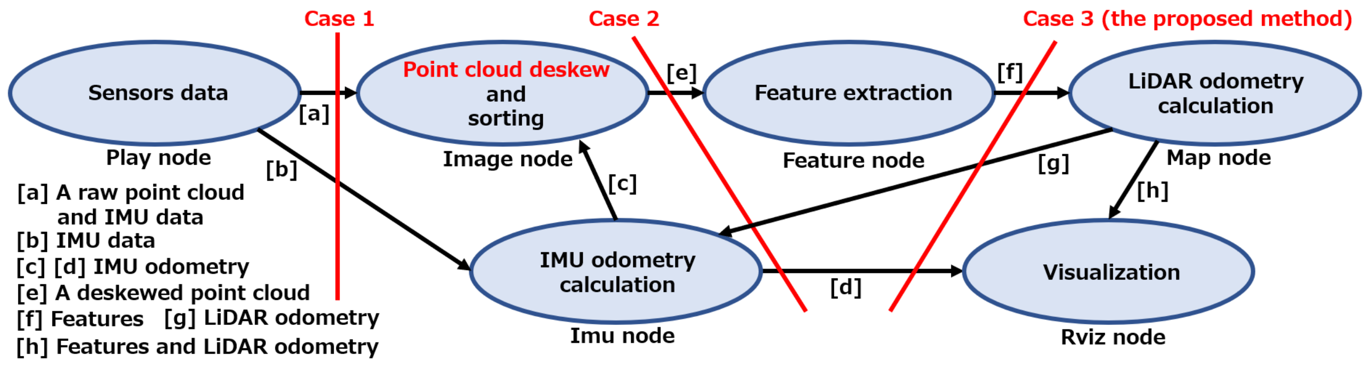
Case 1
Case 2
Case 3 (the proposed method)
Sensors data
Play node
Point cloud deskew and sorting
Image node
Feature extraction
Feature node
LiDAR odometry calculation
Map node
IMU odometry calculation
Imu node
Visualization
Rviz node
[a]
[b]
[c]
[d]
[e]
[f]
[g]
[h]
[a] A raw point cloud and IMU data
[b] IMU data
[c] [d] IMU odometry
[e] A deskewed point cloud
[f] Features [g] LiDAR odometry
[h] Features and LiDAR odometry

Fig. 3. The LIO-SAM nodes

localization inference is likely to degrade because of the incorrect registration results. In LIO-SAM, the point cloud deskew is performed by correcting the rotation angle and position at each scan point.

B. LIO-SAM Algorithm

This section describes the LIO-SAM. Figure 3 shows the ROS nodes along with their tasks in the LIO-SAM implementation. The Play node plays a bag file composed of the LiDAR, IMU, and GPS data and publishes these sensor data to the Image [a], IMU [b] and Map nodes. The Rviz node visualizes results of localization and mapping.

LiDAR odometry $\mathbf{T}_{\mathrm{LiDAR}} \in SE(3)$ is represented as $\mathbf{T}_{\mathrm{LiDAR}} = [\mathbf{R}_{\mathrm{LiDAR}} | \boldsymbol{p}_{\mathrm{LiDAR}}]$ where $\mathbf{R}_{\mathrm{LiDAR}}$ is a rotation matrix and $\boldsymbol{p}_{\mathrm{LiDAR}}$ is a position vector. IMU odometry as $\mathbf{T}_{\mathrm{IMU}} \in SE(3)$ is represented in the same manner. $\mathbf{R}_{\mathrm{LiDAR}}$ is a matrix based on the rotation angles $\boldsymbol{r}_{\mathrm{LiDAR}}$.

1) Imu node: The Imu node holds IMU odometry data $\mathbf{T}_{\mathrm{IMU}}$ and an IMU bias $\boldsymbol{b}$. The Imu node receives IMU data (acceleration and angular velocity data) from the Play node [b] at an IMU frequency and LiDAR odometry from the Map node [g] at an LiDAR frequency. Main roles of the Imu node are odometry ($\mathbf{T}_{\mathrm{IMU}}$) estimation and IMU bias ($\boldsymbol{b}$) estimation. The IMU bias $\boldsymbol{b}$ affects sensor values and needs to be corrected. The odometry estimation is performed by a combination of local factor graph optimization and the latest IMU data. This factor graph is composed of the LiDAR odometry $\mathbf{T}_{\mathrm{LiDAR}}$ and the IMU odometry $\mathbf{T}_{\mathrm{IMU}}$, and this graph optimization is done when IMU node receives the LiDAR odometry $\mathbf{T}_{\mathrm{LiDAR}}$ from Map node. After a certain amount of time, the factor graph is reset because the factor graph grows with time. The optimization result and IMU data are combined, and this process realizes realtime (IMU frequency) odometry $\mathbf{T}_{\mathrm{IMU}}$. This realtime odometry $\mathbf{T}_{\mathrm{IMU}}$ is published to the Image [c] and Rviz nodes [d] at the IMU frequency.

2) Image node: The Image node receives a raw point cloud $\mathcal{P}$ and the IMU data from the Play node [a]. This node also receives the IMU odometry $\mathbf{T}_{\mathrm{IMU}}$ from the IMU node [c]. There are two main roles in the Image node. First, the Image node sets an initial position $\mathbf{T}_{ini}$ for point cloud registration in the Map node. This leads to the faster convergence of the registration because the initial estimate $\mathbf{T}_{ini}$ is sufficiently close to the global optimum. Second, the Image node deskews

Algorithm 1 The funtion including the point cloud deskew

```
Input: Point cloud $\mathcal{P}$, IMU rotation angles $R_{imu}$, IMU times $T_{imu}$, and odometry change $\mathbf{o_{inc}}$
Output: Deskewed point cloud $\mathcal{P}_{deskewed}$, ranges from the LiDAR to points $D$, and column numbers $N_{col}$
1: $\mathbf{M_{inv}}[4][4] \leftarrow$ An inverse matrix of an initial LiDAR pose computed from $\mathcal{P}$, $R_{imu}$, and $\mathbf{o_{inc}}$
2: $t_{start} \leftarrow$ a time of the first point of $\mathcal{P}$
3: $t_{end} \leftarrow$ a time of the end point of $\mathcal{P}$
4: for $i \leftarrow 0, \ldots, 28800$ do
5:     if $i \geq$ size of $P$ then
6:         continue
7:     end if
8:     $\boldsymbol{p} \leftarrow$ the $i$-th point of $\mathcal{P}$
9:     $d \leftarrow$ a range of $\boldsymbol{p}$ from the LiDAR
10:    $id_{row} \leftarrow$ a ring of $p$
11:    $id_{col} \leftarrow$ a column number calculated from $\boldsymbol{p}$ and $id_{row}$
12:    if the position of $\boldsymbol{p}$ is the outside of the matrix || the position on the matrix is the same as another point then
13:        continue
14:    end if
15:    The $i$-th element of $\mathcal{P}_{deskewed} \leftarrow$ a deskewed point computed from $\boldsymbol{p}$, $R_{imu}$, $T_{imu}$, $\mathbf{o_{inc}}$, $t_{start}$, $t_{end}$, and $\mathbf{M_{inv}}$
16:    The $i$-th element of $D \leftarrow d$
17:    The $i$-th element of $N_{col} \leftarrow id_{col}$
18: end for
```

Algorithm 2 Point deskew

```
Input: $p$, $R_{imu}$, $T_{imu}$, $\mathbf{o_{inc}}$, $t_{start}$, $t_{end}$, and $\mathbf{M_{inv}}$
Output: A deskewed point $p_{deskewed}$
1: $t_{\mathrm{LiDAR}} \leftarrow$ a time of the point
2: $t_{\mathrm{IMU_b}} \leftarrow$ an IMU time just before $t_{\mathrm{LiDAR}}$
3: $t_{\mathrm{IMU_a}} \leftarrow$ an IMU time just after $t_{\mathrm{LiDAR}}$
4: $r_{\mathrm{IMU_b}} \leftarrow$ IMU rotation angles at $t_{\mathrm{IMU_b}}$ from $R_{imu}$
5: $r_{\mathrm{IMU_a}} \leftarrow$ IMU rotation angles at $t_{\mathrm{IMU_a}}$ from $R_{imu}$
6: $\boldsymbol{r}_{\mathrm{LiDAR}}[3] \leftarrow$ rotation angles of the LiDAR at $t_p$ calculated from $\boldsymbol{p}$, $t_{\mathrm{LiDAR}}$, $r_{\mathrm{IMU_b}}$, $r_{\mathrm{IMU_a}}$, $t_{\mathrm{IMU_b}}$, and $t_{\mathrm{IMU_a}}$
7: $\boldsymbol{p}_{\mathrm{LiDAR}}[3] \leftarrow$ a position of the LiDAR at $t_{\mathrm{LiDAR}}$ calculated from $\boldsymbol{p}$, $t_{\mathrm{LiDAR}}$, $t_{start}$, $t_{end}$, and $\mathbf{o_{inc}}$
8: $\mathbf{A}[4][4] \leftarrow$ a matrix of the absolute LiDAR pose
9: $\mathbf{B}[4][4] \leftarrow \mathbf{M_{inv}} * \mathbf{A}$    ▷ a matrix of the relative LiDAR pose
10: $p_{deskewed} \leftarrow \mathbf{B}_{0:3,0:3}\boldsymbol{p} + \mathbf{B}_{0:3,3}$
```

valid points within one scan and arranges these points in order. Algorithms for the function for the point cloud deskew are described in Algorithms 1 and 2. Algorithm 1 describes the function for deskewing all the points, and Algorithm 2 describes the funtion only for deskewing each point. Figure 4 shows a flowchart of the point cloud deskew. The next point is deskewed after its range d from the sensor and its column number id_{col} are calculated (Algorithm 1, lines 9 and 11). This

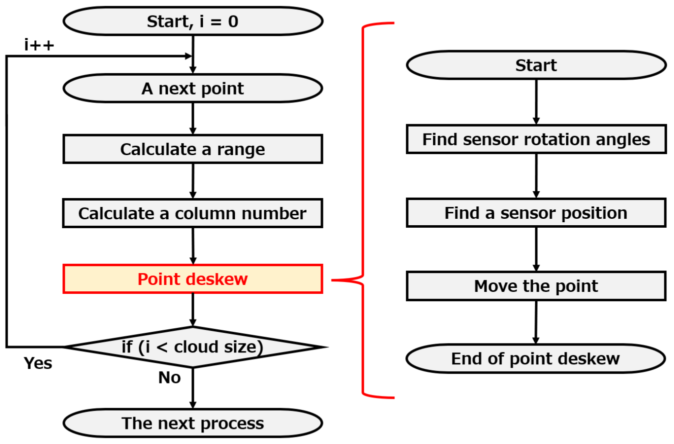

Fig. 4. A point cloud deskew flowchart

range d is used to calculate a curvature for feature extraction in Feature node. The column number id_{col} represents the position of the point within one ring. A sorted point cloud within a scan is formed based on this column number id_{col} and the ring number id_{row}. Without this sorted point cloud, the point cloud registration and the feature extraction cannot be executed.

In the deskew process, the changes in the sensor rotation angles $\boldsymbol{r}_{\text{LiDAR}}$ and translation $\boldsymbol{p}_{\text{LiDAR}}$ relative to the start of a scan need to be computed for each point $\boldsymbol{p}$. The sensor rotation angles $\boldsymbol{r}_{\text{LiDAR}}$ around the x-, y-, and z-axis are calculated by linear interpolation of IMU rotation angles (Algorithm 2, line 6). For example, a rotation angle around the x-axis $\theta_x (=$

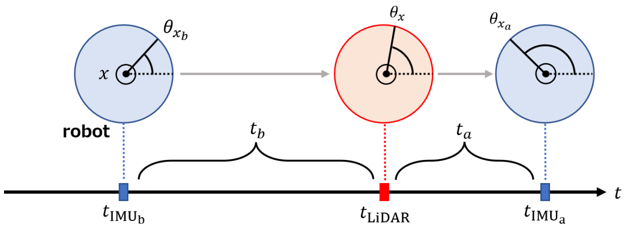

Fig. 5. Linear interpolation for the rotation angle around the x-axis

$\boldsymbol{r}_{\text{LiDAR}}[0])$ is calculated as shown in Figure 5 and as follows (Algorithm 2, lines 1 - 6).

$$\begin{aligned} t_b &= t_{\text{LiDAR}} - t_{\text{IMU}_\text{b}} \\ t_a &= t_{\text{IMU}_\text{a}} - t_{\text{LiDAR}} \\ \theta_x &= \frac{\theta_{x_b} t_a + \theta_{x_a} t_b}{t_a + t_b}, \end{aligned} \tag{1}$$

where t_{LiDAR} is the LiDAR timestamp when a point $\boldsymbol{p}$ is obtained. t_{IMU_b} and t_{IMU_a} are the timestamps of two consecutive IMU data which are acquired right before and after the point is obtained, i.e., $t_{\text{IMU}_\text{a}} \leq t_{\text{LiDAR}} \leq t_{\text{IMU}_\text{b}}$ (Algorithm 2, lines 2 and 3). The sensor translation changes $\mathbf{o}_{\mathbf{inc}}$ are calculated by multiplying the odometry change while obtaining one scan by the ratio of the elapsed time within the scan. A translational change in the x-axis direction x_{trans} is calculated as shown in Figure 6 and as follows (Algorithm 2, line 7).

$$\begin{aligned} r &= t_{\text{LiDAR}} / (t_{end} - t_{start}) \\ x_{trans} &= r \times x_{inc}, \end{aligned} \tag{2}$$

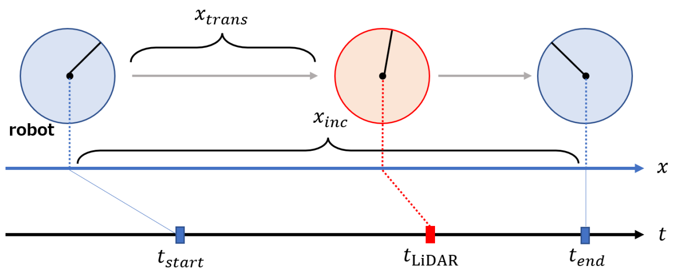

Fig. 6. Linear interpolation for the position in the x-axis direction

where t_{start} and t_{end} are timestamps at which the scan acquisition started and finished. r is the ratio of the elapsed time and x_{inc} is the odometry change while acquiring the scan. x_{inc} $(= \mathbf{o}_{\mathbf{inc}}[\mathbf{0}])$ is based on the two IMU odometry values at t_{start} and t_{end}. This odometry change is calculated from the odometry data from the Imu node. The point is moved based on the rotation angles and the translation changes to deskew the point. A series of these processes is performed for all points in the LiDAR scan. Deskewing the point cloud contributes to the highly accurate localization and mapping. The point cloud deskew is the most expensive process in the Image node. The deskewed point cloud is arranged in order and published to the Feature node [e].

3) Feature node: The Feature node retrieves a point cloud from the Image node [e] and extracts features for each point. The features are classified into two types, edge features and planar features. Points with large curvature are extracted as the edge features. The curvature is calculated simply based on the point distance from the sensor. The remaining points are downsampled and extracted as the planar features. These features are published to the Map node [f].

4) Map node: The Map node is executed when it receives feature data from the Feature node [f]. Main roles of the Map node are point cloud registration and global optimization using a factor graph. Point cloud registration matches the edge and planar features received from the Feature node to each voxel map. These voxel maps are generated by merging multiple preceding scan data and downsampling this merged point cloud. This registration uses the initial location value from the Image node for efficiency. Nearby points from an input point are extracted by KNN and a corresponding edge or plane is generated from these points. The distance of each input point to the corresponding edge or plane is calculated. The sum of the edge distance and the plane distance is used as an objective function as follows.

$$\min_{\mathbf{T}} \left\{ \sum_{p_i^e} d_i^e + \sum_{p_i^p} d_i^p \right\}, \tag{3}$$

where $\mathbf{T}$ is a transformation matrix from LiDAR coordinate system to map coordinate system. p_i^e is the i-th edge feature and d_i^e is the distance from p_i^e to the corresponding edge. p_i^p is the i-th planar feature and d_i^p is also the the distance from p_i^p to the corresponding plane. The position of the point cloud is updated to minimize this objective function by using the Levenberg-Marquardt algorithm. This process is repeated until convergence. The point cloud along with its position is published to the Map node.

The global optimization using a factor graph is executed in the Map node. The factor graph is composed of LiDAR odometry factors, GPS factors, and loop closure factors. LiDAR odometry factors are based on the relative poses between two scans next to each other. GPS factors are based on the absolute location value of GPS. Loop closure factors are based on the loop closure when a robot returns to the previously visited area. The odometry from this optimization is more accurate than that from the graph optimization in the Imu node. The odometry computed from this optimization is published to the Imu node [g] and the Rviz node [h]. The features are also published to the Rviz node [h].

III. Proposed Method

In this section, we propose a partitioning method of LIO-SAM for edge devices and a single server. Moreover, the edge process is accelerated by FPGA to satisfy a realtime constraint.

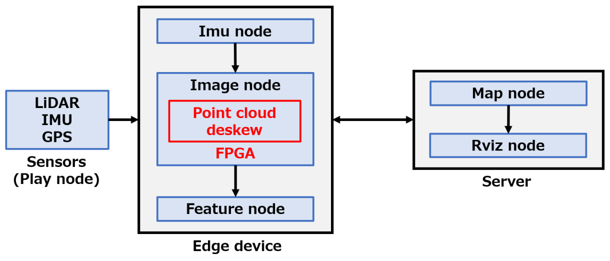

Fig. 7. Proposed edge-server cooperative 3D LiDAR SLAM system

A. Edge-Server Partitioning of LIO-SAM

First, we measure an execution time of Map node on an edge device. In this paper, Xilinx ZCU104 board is used as an edge device (the device specification is described in Section IV). The execution times of Map node range from 500ms to 2500ms. Given that LIO-SAM runs in a 10Hz interval, Map node process is too computationally expensive to be executed in the edge device, and hence should be offloaded to the server. Thus, we need to consider an edge-server cooperative LIO-SAM. In the edge-server cooperative SLAM, data transfer between them is an overhead because a large amount of sensor data should be transferred at high frequency. Reducing the data transfer can alleviate the demands for a high network bandwidth and also a power consumption, which are important for battery-powered edge devices.

We consider several partitioning schemes based on the data transfer size as shown in Figure 3. The data transfer of the edge-server cooperative LIO-SAM consists of point clouds from a 3D LiDAR and IMU data. The size of the point clouds is especially large and the point cloud size needs to be reduced. The feature extraction performed in the Feature node greatly reduces the number of the points, since only a small set of points with salient features are selected and the other points are discarded. We propose to partition the LIO-SAM nodes between Feature node and Map node as shown in Figure 7 and Case 3 of Figure 3. That is, only Map node and Rviz node are executed on the server side. Another benefit of this partitioning scheme is that it does not need to transfer IMU data to the server, which is produced at a high rate of around 500Hz. The advantage of this partitioning method in terms of the data transfer size is evaluated in Section V-C. Specifically, three partitioning cases will be evaluated. In the first case of Figure 3 (Case 1), the edge device transfers raw sensor data to the server, i.e., the server runs the entire LIO-SAM algorithm and the edge device waits for the completion. In the second case (Case 2), the edge device is in charge of running Image and IMU nodes, and transfers the deskewed LiDAR scans to the server. The third case (Case 3) is the proposed method, where the edge device runs Image, Imu, and Feature node, and transfers the feature points to the server.

B. FPGA Acceleration of Edge Process

As shown in Figure 7, the Play node, Imu node, Image node, and Feature node are executed in the edge device. Among these nodes, the Image node and the Feature node take a relatively long time and a route from the Play node to the Feature node via the Image node is a critical path. The LiDAR frequency is set to 10Hz which is the same as the LiDAR scan rate. Given that the Image node and the Feature node are executed in sequence, the total execution time of the Image node and the Feature node must be less than 100ms. IMU execution time is negligible and is run concurrently with the other nodes. It tends to take more than 100ms for the Imu node to receive the LiDAR odometry data from the Map node, but this is not a problem in this paper because it has almost no negative effect on the localization accuracy and the edge execution time. The Imu node can execute accurate realtime localization.

In the edge side, a function for the point cloud deskew in the Image node takes a long time. This function accounts for approximately a half of the execution time on the edge side. Thus, we offload this task to the FPGA fabric to accelerate the edge process. The resultant system is illustrated in Figure 7. In LIO-SAM, a point cloud is deskewed by correcting each point by the amount of sensor pose changes. As mentioned in Section II, the sensor pose changes are calculated from the IMU rotation angles, the odometry, and the timestamps. Each point deskew can be executed independently and thus exhibits a high degree of parallelism. The point deskew processes can be parallelized and pipelined by offloading it to FPGA as shown in Figure 8.

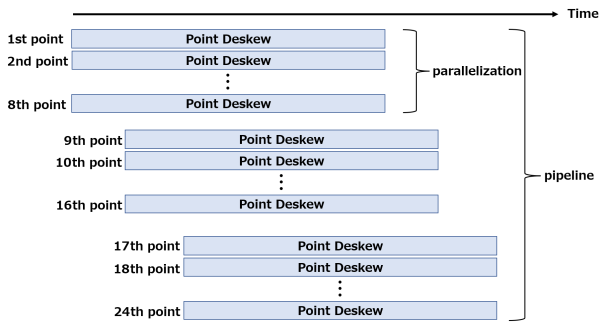

Fig. 8. Parallelism and pipeline of the point cloud deskew

IV. FPGA Implementation

This section describes the FPGA implementation of the point cloud deskew process in detail. We use Xilinx ZCU104 Evaluation Kit (Zynq UltraScale+ MPSoC) as an edge device. Xilinx ZCU104 integrates programmable logic, a quad-core

ARM Cortex-A53 CPU, and 2GB DRAM. Pynq Linux 2.7 based on Ubuntu 20.04 is running on the board. The clock frequency of FPGA is set to 100MHz. We use Xilinx Vitis HLS 2020.2 and Vivado 2020.2 for logic synthesis and place-and-route. The deskew process is implemented on the programmable logic, and the other edge processes are on the ARM Cortex-A53 CPU.

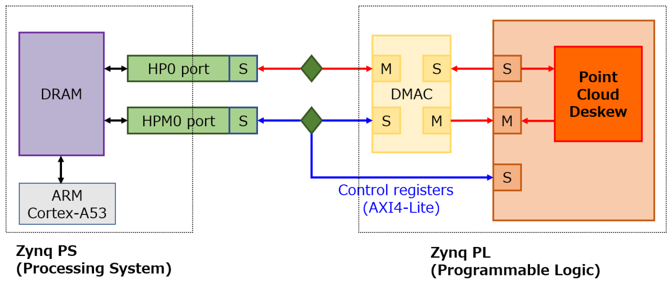
DRAM
HP0 port
S
HPM0 port
S
ARM Cortex-A53
M
S
DMAC
S
M
S
M
S
Point Cloud Deskew
Control registers (AXI4-Lite)
Zynq PS (Processing System)
Zynq PL (Programmable Logic)

Fig. 9. Board level implementation (Xilinx ZCU104 Evaluation Kit)

As shown in Figure 9, a point cloud deskew core and a DMA controller are implemented on Zynq PL (Programmable Logic). The processes before the point cloud deskew are executed on Zynq PS (Processing System), and the point cloud deskew is executed on Zynq PL. The input data including the point clouds and the IMU data is transferred from PL to PS via AXI4-Stream for a fast data transfer. The output data including the deskewed point cloud is sent back from PL to PS. This data is utilized in subsequence processes (e.g., point cloud registration). As mentioned in Section III-B, the point cloud deskew is parallelized and pipelined as shown in Figure 8. The degree of parallelization is set to eight, i.e., eight points are deskewed in parallel, the latency between the preceding and following parallel processes is 0.18 μs.

In this paper, we use an implementation of LIO-SAM [2] from the authors of [1], but modify the point cloud deskew function to use our custom FPGA-based accelerator. The point data type is changed to contain xyz coordinates, a ring number, and a timestamp for reducing unnecessary data transfer and resource consumption. Processes that are difficult to be implemented on PL part in the function for the deskew are left to PS part.

V. Evaluations

In this section, we evaluate the proposed edge-server cooperative 3D LiDAR SLAM based on the FPGA implementation in terms of FPGA resource utilization, data transfer size, execution time of the edge process, and execution time including communication time.

A. Dataset

We use three datasets which are used in the evaluation of LIO-SAM [1]. These datasets contain point cloud from a Velodyne VLP-16 LiDAR, acceleration and angular velocity from a MicroStrain 3DM-GX5-25 IMU, and absolute location from a Reach M GPS. The number of channels of Velodyne VLP-16 is 16, and the number of the points within on ring is 1800. The maximum point cloud size in one scan is 28800 (16 $\times$ 1800). These datasets are named Walking Dataset, Campus Dataset, and Park Dataset. The details about the datasets are summarized in Table I.

[2]https://github.com/TixiaoShan/LIO-SAM

TABLE I
Details about Datasets

Dataset	Scans	Elevation change (m)	Trajectory length (m)	Max rotation speed ($^\circ$ /s)
Walking	6502	0.3	801	133.7
Campus	9865	1.0	1437	124.8
Park	24691	19.0	2898	217.4

B. FPGA Resource Utilization

FPGA resource utilization for the implementation of the point cloud deskew is shown in Table II. As shown, the utilization is less than 50% of the total resources available on Xilinx ZCU104 board for all the primitives. The results indicate the following three possibilities. First, we can use a higher resolution 3D LiDAR which produces LiDAR scans with more points. The number of channels assumed in this evaluation is 16, and there are finer-grained 3D LiDARs that have 32 or 64 channels. Second, we can consider further acceleration by implementing more processes on FPGA. For example, we can accelerate feature extraction in the Feature node by FPGA. However, the FPGA acceleration of feature extraction is not effective to reduce execution time, because it does not take a time as much time as the point cloud deskew. Third, it is possible to increase the number of data transfer ports between PS and PL parts for further acceleration because data transfer time between them accounts for about 80% of execution times on FPGA.

TABLE II
FPGA Resource Utilization (after place and route)

	BRAM	URAM	DSP	FF	LUT
Total	71.5	40	327	53687	56242
Available	312	96	1728	460800	230400
Utilization	22.92%	41.67%	18.92%	11.65%	24.41%

C. Data Transfer Size

This section evaluates the data transfer size for three partitioning schemes. Specifically, we evaluate three partitioning cases shown in Figure 3. We take point clouds and IMU data into account for this data transfer size evaluation. The data sizes are calculated by taking the average of 6000 to 25000 scan processes in each dataset.

Table III shows data transfer sizes of the three partitioning cases for the Walking Dataset. The data transfer size of Case 2 is 1.02 times larger than that of Case 1 because the number of points is almost equal but data size for each point is increased by transferring new data, such as its range and its column number. This means that executing the Image node and Imu node in the edge while executing the Feature node in the server is not as effective. The data transfer size of Case 3 (our proposal) is 6.67 times smaller than that of Case 1. Case 3 reduces the data transfer significantly because the number of the points is reduced to about one-fourth by the feature extraction. Case 3 also eliminates the need to send IMU data to the server. These results demonstrate that the Feature node should be executed in the edge by considering the data transfer size for the edge-server cooperative LIO-SAM.

The data transfer size of Case 3 for the three datasets are shown in Table IV. The significant data transfer size

TABLE III
DATA TRANSFER SIZE OF WALKING DATASET

Partitioning method	Points (Mbps)	IMU (Mbps)	Total (Mbps)	Comparison with Case 1
Case 1	38.39	1.38	39.77	1.0
Case 2	40.49	0.0	40.49	1.02
Case 3	5.90	0.0	**5.90**	**0.15**

reduction is observed in all the datasets, and the data transfer is reduced by at least a quarter. Reducing data transfer can relax a network constraint for realtime SLAM, and reduce the communication fee and the energy consumption.

TABLE IV
DATA TRANSFER SIZE OF THREE DATASETS

Dataset	Case 3 data transfer size (Mbps)	Comparison with Case 1
Walking	5.90	0.15
Campus	7.64	0.23
Park	7.23	0.18

D. *Execution Time with FPGA Acceleration*

LiDAR frequency is set to 10Hz which is a typical value in LiDAR SLAM; in this case, the execution time of the edge should not exceed 100ms. We assume that the execution time on the edge side is a sum of execution times of Image node and Feature node which process the LiDAR scans. The execution times of the Imu node are small enough to be ignored.

The execution time on the edge side of each point cloud scan for Walking Dataset is shown in Figure 10. The orange and blue bars represent execution times with and without FPGA acceleration. A few scan data exceeds the 100ms constraint and an average execution time becomes larger when the FPGA acceleration is not applied. By implementing the point cloud deskew on the FPGA, all the scans are processed within 100ms (i.e., the SLAM process does not violate the realtime constraint) and scan processes become faster as a result. Please note that not many scans exceed the 100ms constraint without the FPGA acceleration, and these scans may not affect SLAM performance significantly in this dataset. However, it is important not to lose scan data especially when the sensor is moving fast and the environmental map is changing rapidly. When a higher resolution LiDAR is used, the execution times on the edge side become longer. In this case, it is more likely to violate the 100ms constraint, and the benefit of the FPGA acceleration increases.

Tables V and VI show the execution times on the edge side with and without the FPGA acceleration. Table V shows execution times of the point cloud deskew, and Table VI shows that of the entire edge process. Figure 11 shows a histogram of the execution time on the edge side. As shown in Table V, the execution time of the point cloud deskew which accounted for approximately half of the edge execution time is reduced to 7.07ms. The FPGA implementation of the point cloud deskew is 3.46 to 4.31 times faster than the software execution on average. It is up to 10.02-13.23 times faster than the software counterpart. As shown in Table VI, an entire execution time on the edge side is 1.63 to 1.73 times shorter than that without the FPGA acceleration on average. The worst case (the maximum) execution time is also reduced from over 100ms to within 100ms. That is, the execution time constraint on the edge side is met by the proposed FPGA acceleration.

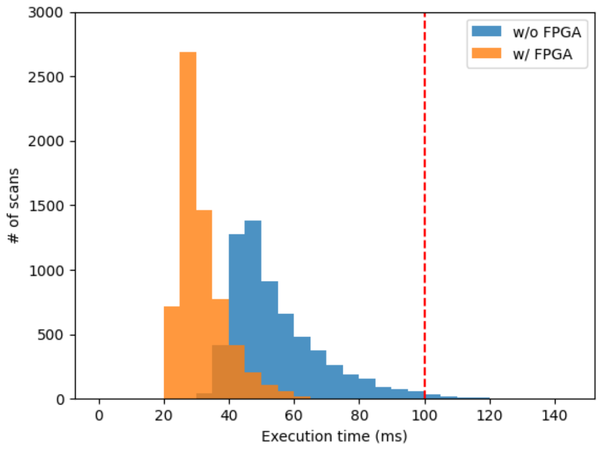

Fig. 10. The distribution of the execution time in Walking Dataset

TABLE V
EXECUTION TIME OF THE POINT CLOUD DESKEW (MS)

Dataset	Average, standard deviation	Maximum	With FPGA
Walking	30.03 ± 10.15	81.23	7.07
Campus	24.43 ± 8.14	70.86	7.07
Park	30.48 ± 10.58	93.58	7.07

TABLE VI
AVERAGE AND MAXIMUM EXECUTION TIME ON THE EDGE SIDE (MS)

Dataset	Average w/o FPGA	Average w/ FPGA	Maximum w/o FPGA	Maximum w/ FPGA
Walking	54.72	31.76	147.51	80.27
Campus	45.00	27.64	109.73	67.76
Park	55.69	32.28	138.91	83.80

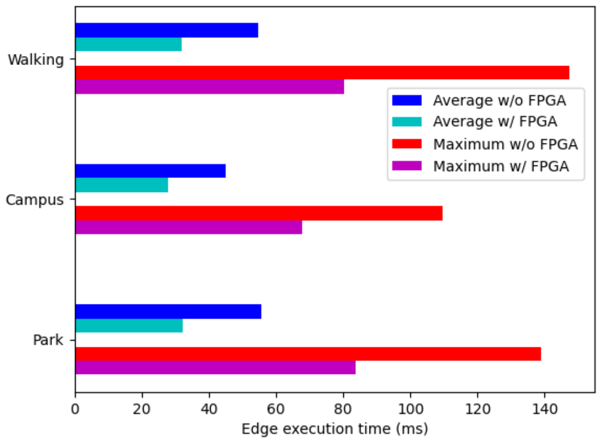

Fig. 11. Average and maximum execution times on the edge side

E. *Execution Time Including Communication Overhead*

In this section, we evaluate the total execution time of edge process including the data transfer from/to the server. This total time is a sum of execution times of the Image node, the Feature node, and the communication overhead estimated from the given network bandwidth. The execution time of the Map node on the server is not considered in this evaluation as it only accounts for a negligible part of the total execution time. If the edge-server cooperative SLAM can use an ideal network and the communication time is negligible, we do not have to consider the communication overhead. However, when multiple SLAM systems work or the network bandwidth is narrow, the communication time increases.

Figure 12 shows the total execution time including the communication time at various bandwidths. As shown in Table III, Case 1 transfers a raw point cloud to the server, and thus the communication time is larger than that of Case 3. Case 1 is the best method in terms of the execution time if the network bandwidth is sufficiently high. However, the benefit of Case 3 increases when the bandwidth is narrow as shown in Figure 12. In Case 3 without the FPGA acceleration, the proposed edge-server cooperative SLAM is beneficial if a network bandwidth is less than 67.85Mbps. In the case of the FPGA acceleration, the proposed method is beneficial if a network bandwidth is less than 125.63Mbps.

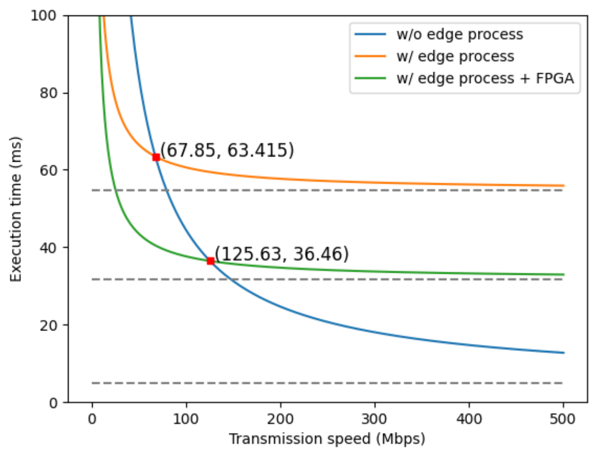

Fig. 12. Total time of the Image node, the Feature node, and the communication in Walking Dataset

VI. Summary

It is a challenging problem to meet the realtime constraint of SLAM algorithms due to their high computational cost when executed solely on the edge devices. In this paper, we proposed an edge-server cooperative 3D LiDAR SLAM based on the state-of-the-art method, LIO-SAM. Our proposed method reduces the computational complexity on the edge side, thereby allowing to run multiple SLAM instances or the other tasks such as path planning on a single server. We partitioned tasks in the LIO-SAM to the edge and server so that the data transfer between them is reduced. In this paper, we adopted a partitioning strategy, where the edge device performs the feature extraction and the server performs a global optimization. By only transferring a subset of points with salient features to the server, the amount of data transfer is reduced by 4.35 to 6.67 times compared to transferring the raw LiDAR scans. Moreover, we accelerate the edge process by offloading the point cloud deskew to the programmable logic part of the Xilinx ZCU104 board. The execution time of the edge process with the proposed FPGA accelerator is 1.63 to 1.73 times shorter than without FPGA. Furthermore, the execution times of all the scans are reduced to less than 100ms, meaning that the proposed method met the execution time constraint. As a future work, we are considering using different SLAM algorithms, different 3D LiDAR sensors, and power consumption in real environments.

Acknowledgments This work was partially supported by JST AIP Acceleration Research JPMJCR23U3 and JSPS KAKENHI Grant Numbers JP22H03596, Japan.

References

[1] T. Shan *et al.*, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Jul. 2020, pp. 5135–5142.

[2] J. Zhang and S. Singh, "LOAM : Lidar Odometry and Mapping in real-time," in *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Jan. 2014, pp. 109–111.

[3] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[4] H. Ye, Y. Chen, and M. Liu, "Tightly Coupled 3D Lidar Inertial Odometry and Mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3144–3150.

[5] M. Karrer, P. Schmuck, and M. Chli, "CVI-SLAM–Collaborative Visual-Inertial SLAM," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, Oct. 2018.

[6] H. Ye, Y. Chen, and M. Liu, "A Collaborative Visual SLAM Framework for Service Robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2021.

[7] E. B. Olson, "Real-Time Correlative Scan Matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2009, pp. 4387–4393.

[8] K. Sugiura and H. Matsutani, "A Universal LiDAR SLAM Accelerator System on Low-Cost FPGA," *IEEE Access*, vol. 10, pp. 26 931–26 947, Mar. 2022.

[9] M. Flottmann, M. Eisoldt, J. Gaal1, M. Rothmann, M. Tassemeier, T. Wiemann, and M. Porrman, "Energy-efficient FPGA-accelerated LiDAR-based SLAM for Embedded Robotics," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT)*, Dec. 2021, pp. 1–6.

[10] H. Sun, X. Liu, Q. Deng, W. Jiang, S. Luo, and Y. Ha, "Efficient FPGA Implementation of K-Nearest-Neighbor Search Algorithm for 3D LIDAR Localization and Mapping in Smart Vehicles," *IEEE Transactions on Circuits and Systems II*, vol. 67, no. 9, pp. 1644–1648, Aug. 2020.

[11] K. Sugiura and H. Matsutani, "P3Net: PointNet-based Path Planning on FPGA," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT)*, Dec. 2022.

[12] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 652–660.

[13] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 14, no. 2, pp. 239–256, Feb. 1992.

[14] P. Biber and W. Straßer, "The Normal Distributions Transform: A New Approach to Laser Scan Matching," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 2743–2748.

[15] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2022.

[16] X. Zheng and J. Zhu, "Efficient LiDAR Odometry for Autonomous Driving," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8458–8465, Oct. 2021.

[17] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li, "LO-Net: Deep Real-time Lidar Odometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8473–8482.

[18] K. Li, M. Li, and U. D. Hanebeck, "Towards High-Performance Solid-State-LiDAR-Inertial Odometry and Mapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5167–5174, Jul. 2021.