A Unified Accelerator Design for LiDAR SLAM Algorithms for Low-end FPGAs

Keisuke Sugiura and Hiroki Matsutani
Dept. of ICS, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: {sugiura,matutani}@arc.ics.keio.ac.jp

2021 International Conference on Field-Programmable Technology (ICFPT) | 978-1-6654-2010-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICFPT52863.2021.9609886

***Abstract*—A fast and reliable LiDAR (Light Detection and Ranging) SLAM (Simultaneous Localization and Mapping) system is the growing need for autonomous mobile robots, which are used for a variety of tasks such as indoor cleaning, navigation, and transportation. To bridge the gap between the limited processing power on such robots and the high computational requirement of the SLAM system, in this paper we propose a unified accelerator design for 2D SLAM algorithms on resource-limited FPGA devices. As scan matching is the heart of these algorithms, the proposed FPGA-based accelerator utilizes scan matching cores on the programmable logic part and users can switch the SLAM algorithms to adapt to performance requirements and environments without modifying and re-synthesizing the logic part. We integrate the accelerator into two representative SLAM algorithms, namely particle filter-based and graph-based SLAM. They are evaluated in terms of resource utilization, processing speed, and quality of output results with various real-world datasets, highlighting their algorithmic characteristics. Experiment results on a Pynq-Z2 board demonstrate that scan matching is accelerated by 13.67–14.84x, improving the overall performance of particle filter-based and graph-based SLAM by 4.03–4.67x and 3.09–4.00x respectively, while maintaining the accuracy comparable to their software counterparts and even state-of-the-art methods.**

***Index Terms*—SLAM, Scan Matching, FPGA**

I. Introduction

LiDAR (Light Detection and Ranging) SLAM (Simultaneous Localization and Mapping) is the task for creating a precise, globally consistent map of a surrounding environment and estimating a robot pose inside the map using a sequence of LiDAR scans (Figure 1). SLAM plays a central role for positioning systems on autonomous robots especially in GPS-denied scenarios, and becomes the basis of many useful applications such as house cleaning, navigation in offices, and transportation in warehouses. Though LiDAR SLAM is a relatively mature and well-established technique, it requires high-end CPUs and even GPU accelerations due to the high computational complexity of the state-of-the-art methods [1]–[5]. This hinders the use of SLAM in indoor mobile robots, due to their low computational capability and limited power supply. Developing an embedded FPGA-based SLAM system that achieves both energy efficiency and real-time performance is the promising solution to tackle this problem, and should bring significant benefits to both industry and household.

A number of FPGA-based SLAM accelerators have been proposed in the literature [6]–[12]. Boikos *et al.* [9] proposes the accelerator for direct tracking in LSD-SLAM, which estimates a camera pose by minimizing per-pixel intensity differences between two images. Their implementation using Zedboard (Xilinx Zynq-7020 SoC) achieves 22.7fps at the input resolution of 320x240 px, which is the 10x speedup compared to the dual-core ARM Cortex-A9 CPU. Gautier *et al.* [10] implements the depth fusion and ray casting parts in InfiniTAM algorithm on Terasic DE1 SoC (Cyclone V), and achieves the 3.16x speedup (0.49 to 1.55fps) at 320x180 compared to Cortex-A9. Liu *et al.* [11] devises the rotationally-invariant hardware-friendly feature descriptor and accelerates the feature detection and matching in ORB-SLAM using Xilinx Zynq XCZ7045 SoC (Kintex 7), resulting in 31x speedup at 640x480 compared to Cortex-A9. These researches focus on Visual SLAM methods which use cameras as the primary sensor. LiDAR is also predominantly employed in SLAM and provides robust, accurate, wide field-of-view information about the surrounding environment, enabling reliable SLAM systems [13] [14]. Despite the importance and widespread use in both industry and household, only a few studies have considered the hardware acceleration of LiDAR SLAM to address its high computational complexity and memory footprint.

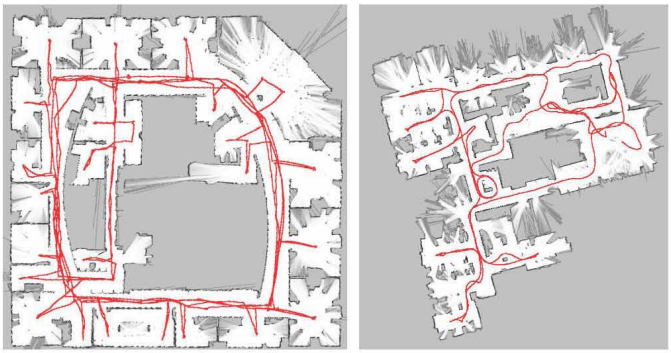

Fig. 1. Grid map and robot trajectory (red line) obtained using our proposed FPGA-based SLAM accelerator (left: particle filter-based SLAM and **Intel** Research Lab dataset, right: graph-based SLAM and **Revo LDS** dataset)

In this paper, we propose an efficient accelerator design for 2D LiDAR SLAM methods targeting low-cost FPGA SoC (Figure 4). Based on the profiling results, we choose to implement the scan matching part on the FPGA fabric to maximize the overall system performance. Our design is modular in a way that it can be integrated to various LiDAR SLAM methods, which allows users to choose the most suitable one according to performance requirements and environmental characteristics. Specifically, we develop an

accelerator for Correlative Scan Matching (CSM) proposed by Olson *et al.* [15], since it strikes a balance between computational efficiency and simplicity. To parallelize the algorithm and further improve the efficiency, we modify the original CSM algorithm and also perform design optimizations, such as data reordering and loop interchanging. The proposed design is implemented using Pynq-Z2 board and applied to two representative SLAM approaches: particle filter-based and graph-based SLAM. Experimental results using real-world datasets confirm that the performance of both methods is effectively boosted while maintaining the accuracy, and also highlight the trade-off between computational cost and accuracy in these methods, which would support our unified design concept.

The novelty of this paper lies in (1) deriving a hardware-friendly robust scan matching algorithm, and (2) proposing a unified accelerator design which is applicable to various 2D LiDAR SLAM methods. This paper is outlined as follows: the next section presents a brief explanation of the two SLAM approaches mentioned above along with scan matching. Modifications to CSM algorithm and design optimizations are described in Section III. Section IV illustrates the implementation details and shows the evaluation results in terms of resource utilization, power consumption, throughput, and quality of outputs. Section V concludes this paper.

II. Preliminaries

SLAM methods are often divided into two main categories: particle filter-based SLAM (PF-SLAM) and graph-based SLAM, which are described in the following sections.

A. Particle Filter-based SLAM (PF-SLAM)

PF-SLAM [16]–[18] uses a set of particles, each of which carries a hypothesis about robot trajectory and map. To filter out particles with less probable hypotheses, importance weight is computed by performing scan matching for each particle. Scan matching is the process to estimate particle's pose by maximizing the overlap between particle's map and scan, as shown in Figure 2. It is the most suitable candidate for hardware acceleration, as it usually becomes a bottleneck, and the computation for each particle is independent, i.e., completely parallelizable [19]–[22].

B. Graph-based SLAM

In graph-based SLAM [23]–[25] pose graph is used as a central data structure, whose node represents a robot pose and edge defines a relative pose between two nodes. Edges represent either odometry or loop constraint: the former is incrementally created in the frontend scan matching, where a new scan is aligned with the latest submap to estimate the current robot pose (submap is created from a series of scans within a certain time window). The latter is inserted after a successful loop detection in backend, which involves scan matching between recent scans and old submaps to detect whether a robot is revisiting previously explored areas. Backend then optimizes the pose graph to refine the entire trajectory estimate. Graph-based SLAM only maintains the most plausible estimate of trajectory and map, and explicitly performs loop detections to improve accuracy. This contributes to the reduced computational cost compared to PF-SLAM. As explained above, both frontend and backend rely heavily on scan matching, which motivates the hardware acceleration of it in graph-based SLAM as well as PF-SLAM [26]–[28].

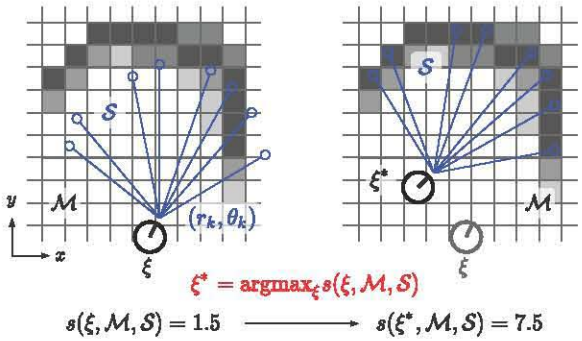

Fig. 2. Updating a robot pose $\boldsymbol{\xi}$ by scan matching. Blue circles represent scan points projected onto a map $\mathcal{M}$, and squares filled with darker color indicate grid cells with higher occupancy probabilities. Scan matching aligns a scan $\mathcal{S}$ with a map $\mathcal{M}$ and seeks a robot pose $\boldsymbol{\xi}^*$ (right), by maximizing a score $s(\boldsymbol{\xi}, \mathcal{M}, \mathcal{S})$. The score is obtained by summing up occupancy probabilities of grid cells which correspond to scan points. If a scan is successfully aligned, an occupancy probability for each scan endpoint is closer to one (blue circles lie on darker cells), resulting in a higher score.

C. Scan-matching

The aim of scan matching is to find a scan pose $\boldsymbol{\xi}^*$ which maximizes a matching score $s(\boldsymbol{\xi}, \mathcal{M}, \mathcal{S})$. Score correlates to the quality of an alignment between map $\mathcal{M}$ and scan $\mathcal{S}$ under the current pose estimate $\boldsymbol{\xi}$. Scan is a set of observed points $\mathcal{S} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, where each point $\mathbf{z}_i$ is defined by a range and an angle (r_i, θ_i) from the sensor origin. Grid map $\mathcal{M}$ partitions the environment into equally-sized grid cells with a resolution of r, each of which stores a probability that it is occupied by an obstacle. We denote the occupancy probability at a cell (i, j) as $\mathcal{M}(i, j)$. Score is evaluated by projecting scan points onto grid cells and summing up their associated occupancy probabilities, as depicted in Figure 2.

In scan matching, score s needs to be evaluated for each solution candidate $\boldsymbol{\xi}$, which involves coordinate transformations for every scan point and memory accesses to grid cells. Memory access patterns for grid maps are usually irregular and unpredictable, since there is no assumption for the shape of input scans. Thus, scan matching is both compute- and data-intensive process and becomes a major bottleneck in SLAM applications. We aim to address this by FPGA-based scan matching acceleration and enable SLAM on low-power edge devices. In the following section, we describe the Correlative Scan Matching (CSM) algorithm [15] and highlight its advantages in terms of hardware efficiency compared to other existing methods.

D. Correlative Scan Matching (CSM)

Algorithm 1 summarizes the CSM algorithm [15]. From a discrete search window of size $(2w_x, 2w_y, 2w_\theta)$, CSM finds the best solution $(n_x^*, n_y^*, n_\theta^*)$ and obtains the scan pose $\boldsymbol{\xi}^*$ (line 16). For any candidate (n_x, n_y, n_θ), its associated pose

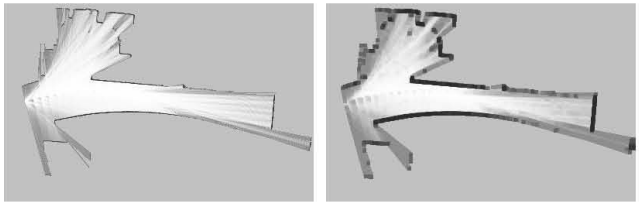

Fig. 3. Example of **fine** $\mathcal{M}$ (left) and **coarse** $\mathcal{M}'$ (right) grid maps

is computed as $\boldsymbol{\xi}^0 + [r \cdot n_x, r \cdot n_y, \delta_\theta \cdot n_\theta]$, where r and δ_θ denote the step sizes along x, y and θ directions. $\boldsymbol{\xi}^0$ is the pose at the search window center (i.e., $n_x, n_y, n_\theta = 0$).

The algorithm is basically a special case of Branch-and-Bound (BB) [29] [1] and utilizes two-level resolution grid maps (i.e., coarse map $\mathcal{M}'$ and fine map $\mathcal{M}$, Figure 3). The coarse map $\mathcal{M}'$ is computed from the input map $\mathcal{M}$ using the **sliding window maximum** operation (line 1) as follows:

$$\mathcal{M}'(i,j) = \max_{i',j' \in [0,1,\ldots,w-1]} \mathcal{M}(i+i', j+j'). \quad (1)$$

Each cell in $\mathcal{M}'$ contains a local maximum within a corresponding small region (of $w \times w$ cells) in $\mathcal{M}$.

The outermost loop (line 4) iterates over θ. For each angle n_θ, CSM projects scan points onto grid cells and computes the set of discrete indices $\mathcal{I} = \{(i_1, j_1), \ldots, (i_N, j_N)\}$ based on the pose $\boldsymbol{\xi}^0 + [0, 0, \delta_\theta \cdot n_\theta]$. CSM then performs **coarse matching** (lines 6-10) using $\mathcal{M}'$, i.e., it evaluates scores s' for all points (n'_x, n'_y, n_θ) in a coarse search grid:

$$s'(n'_x, n'_y, n_\theta) = \sum_{k=1}^{N} \mathcal{M}'(i_k + n'_x, j_k + n'_y). \quad (2)$$

Substitution of (1) into (2) reveals that the score $s'(n'_x, n'_y, n_\theta)$ is the upper-bound of scores (Equation 3) in the $w \times w$ search space starting from (n'_x, n'_y). This allows the pruning at line 10 and improves the algorithm efficiency. As a result, CSM is able to identify the region of interest that contains a global optimum $\boldsymbol{\xi}^*$, and rule out a large part of the search space.

In **fine matching** (lines 11-15), a score s for each point (n_x, n_y, n_θ) in the $w \times w$ region starting from (n'_x, n'_y, n_θ) is evaluated using the fine map $\mathcal{M}$, and the current solution $(n^*_x, n^*_y, n^*_\theta)$, s^* is updated:

$$s(n_x, n_y, n_\theta) = \sum_{k=1}^{N} \mathcal{M}(i_k + n_x, j_k + n_y). \quad (3)$$

Throughout the paper, we set to $r = 5\text{cm}$ and $w = 8$. In this setting, coarse matching sweeps the entire search space at $r \cdot w = 40\text{cm}$ resolution, and fine matching tries to find the optimal solution from $w^2 = 64$ candidate points equally spaced 5cm apart inside the area of size $40 \times 40\text{cm}$. As apparent in line 16, CSM estimates translational and rotational components of the pose $\boldsymbol{\xi}^*$ at $r = 5\text{cm}$ and δ_θ rad accuracy.

As shown above, CSM is the simplified version of BB and is classified as a non-iterative method. Unlike BB counterparts, which use multi-resolution maps and tree data structures, CSM only requires one map $\mathcal{M}'$ to be precomputed and works without any complex data structure as evident in Algorithm 1. This reduces the preprocessing overhead and resource consumptions in exchange for a slight efficiency loss. Compared to iterative methods like hill-climbing [30] or Gauss-Newton [31] [32], which tend to be trapped in local optima, CSM does not depend on initial guesses and guarantees the global optimality of the solution. From the above considerations, we choose to use CSM in our implementation, since it is robust, hardware-friendly, and strikes a balance between algorithm simplicity and efficiency. We describe the design and implementation of our FPGA-based CSM core in the next section.

Algorithm 1 Correlative Scan Matching (CSM)

Require: $\mathcal{M}$, $\mathcal{S}$, $\boldsymbol{\xi}^0$, (w_x, w_y, w_θ), r, δ_θ, w
Ensure: Optimal pose $\boldsymbol{\xi}^*$, score s^*
1: Compute coarse map $\mathcal{M}'$ from $\mathcal{M}$ (Eq. 1)
2: $s^* \leftarrow -\infty$, $(n^*_x, n^*_y, n^*_\theta) \leftarrow (-w_x, -w_y, -w_\theta)$
3: Compute $\widehat{w}_x, \widehat{w}_y$ such that $2w_x = w \cdot \widehat{w}_x, 2w_y = w \cdot \widehat{w}_y$
4: **for** $n_\theta = -w_\theta, \ldots, w_\theta$ **do**
5: Compute indices $\mathcal{I}$ from $\mathcal{S}$ and $\boldsymbol{\xi}^0 + [0, 0, \delta_\theta \cdot n_\theta]$
▷ Coarse matching
6: **for** $n'_y = -w_y, -w_y + w, \ldots, -w_y + (\widehat{w}_y - 1)w$ **do**
7: **for** $n'_x = -w_x, -w_x + w, \ldots, -w_x + (\widehat{w}_x - 1)w$ **do**
8: $s' \leftarrow \sum_{k=1}^{N} \mathcal{M}'(i_k + n'_x, j_k + n'_y)$ (Eq. 2)
9: **if** $s' \leq s^*$ **then**
10: **Continue**
▷ Fine matching
11: **for** $n_y = n'_y, \ldots, n'_y + w - 1$ **do**
12: **for** $n_x = n'_x, \ldots, n'_x + w - 1$ **do**
13: $s \leftarrow \sum_{k=1}^{N} \mathcal{M}(i_k + n_x, j_k + n_y)$ (Eq. 3)
14: **if** $s > s^*$ **then**
15: $s^* \leftarrow s$, $(n^*_x, n^*_y, n^*_\theta) \leftarrow (n_x, n_y, n_\theta)$
16: **return** $s^*, \boldsymbol{\xi}^* = \boldsymbol{\xi}^0 + [r \cdot n^*_x, r \cdot n^*_y, \delta_\theta \cdot n^*_\theta]$

III. Design Optimization

A. Overview of the CSM Core Design

Figure 4 illustrates the block diagram of our board-level implementation. The programmable logic (PL) part contains two CSM cores, each of which independently performs CSM algorithm upon the request from the Zynq processing system (PS). CSM is performed by the following three steps: (1) PS places the input data (scan $\mathcal{S}$ and map $\mathcal{M}$) into the contiguous DRAM buffer. (2) PS then transfers the input to the CSM core via Direct Memory Access (DMA) and starts the CSM core. (3) After the completion, PS receives the result (pose $\boldsymbol{\xi}^*$ and matching score s^*) through the DMA and passes them to the downstream SLAM modules. Importantly, the design is independent from the choice of underlying 2D LiDAR SLAM methods, and users can switch to different suitable methods (e.g., PF-SLAM and graph-based SLAM) to adapt to the environments and performance requirements without modifying and re-synthesizing the design.

Our design conforms to the AXI4-Stream protocol and uses one 64-bit high-performance (HP) port for each DMA controller for the high-speed burst transfer between DRAM and on-chip block RAM (BRAM) (blue arrows in Figure 4). The control registers in DMA controllers and CSM cores

serve as the interface to the PS (red arrows in Figure 4). PS writes to these registers through the AXI4-Lite protocol and memory-mapped I/O to configure the physical address range of the buffer to be transferred, and to specify the algorithmic parameters of CSM (e.g., w_x, w_y, w_θ, w in Algorithm 1).

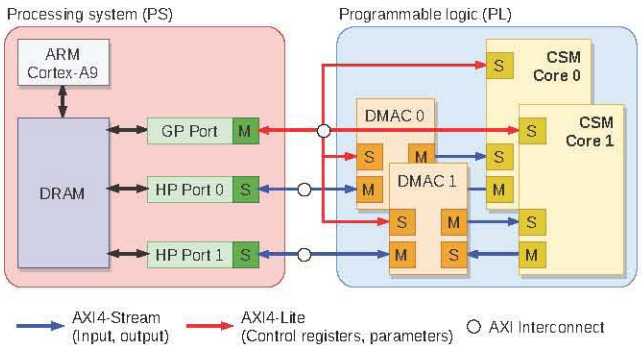

Fig. 4. Block diagram of our implementation

B. Details of the CSM Core Design

Our CSM core is comprised of a set of modules: (i) main controller, (ii) sliding window maximum, (iii) floating-point to fixed-point, (iv) optimizer, (v) scan discretization, (vi) coarse matching, and (vii) fine matching. Figure 6 illustrates the data flow and interactions between these modules inside the core. The functionality of each module is outlined as follows: Module (i) reads the incoming data and configurations and controls the other modules.

Module (ii) retrieves the quantized grid map values $\mathcal{M}$ from the AXI4-Stream interface and stores them to the BRAM buffer (Fine map in Figure 6). It simultaneously applies a sliding window maximum filter (Equation 1) to the grid map values $\mathcal{M}$ to obtain a low-resolution grid map $\mathcal{M}'$, which is stored to another BRAM buffer (Coarse map in Figure 6). The module retrieves map values in a row-wise order, stores column-wise maxima to a temporary cache (of size 320 columns $\times$ w rows), and then computes $\mathcal{M}'$ by taking row-wise maxima of values stored in the cache. In our design, each 64-bit data packet coming from the streaming interface contains eight consecutive map values in 8-bit integer format as shown in Figure 5. Module (iii) extracts two 32-bit floating-point values (range r and angle θ of a scan point $\mathbf{z}$) from a 64-bit data packet (Figure 5), converts them to the 32-bit fixed-point values, and fills the scan buffer. The scan buffer (Figure 6) stores up to 512 range-angle pairs, i.e., $N \leq 512$.

The optimizer module (iv) manages the entire process of the CSM (Algorithm 1) using submodules (v)-(vii). For a given orientation n_θ, the discretization module (v) computes the grid cell indices corresponding to scan points (denoted as $\mathcal{I}$ in Section II-D) and writes them back to the indices buffer. The coarse (vi) and fine (vii) matching modules evaluate solution candidates (Equations 2, 3) in parallel using the discretized scan indices and grid maps $\mathcal{M}, \mathcal{M}'$ located in the BRAM buffer. After the matching, the optimizer module (iv) returns back the optimal solution n_x^*, n_y^*, n_θ^* along with the score

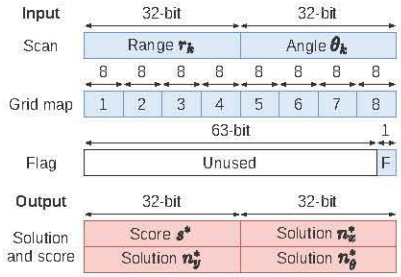

Fig. 5. Input and output packet formats for data transfer between PS-PL

s^* through the AXI4-Stream interface (Figure 5). The following Sections III-C-III-F describe the design optimizations employed to exploit the inherent parallelism of CSM and realize the implementation on resource-constrained low-power FPGAs.

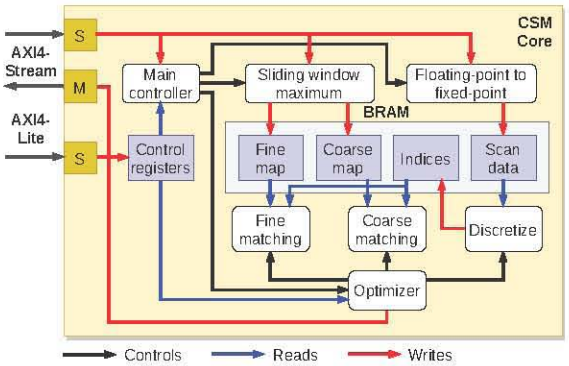

Fig. 6. Block diagram of the CSM core

C. Tuning the Grid Map Buffer Size

In the matching step (lines 8, 13 in Algorithm 1), the number of grid map accesses is $2\widehat{w}_x\widehat{w}_y w_\theta \left(1+w^2\right) N$ in the worst case, which reaches tens of millions in a typical setting. In addition to this, as mentioned in Section II-C, score evaluations (Equations 2, 3) exhibit irregular and unpredictable memory access patterns, which decrease the effectiveness of prefetching grid map values. Reading only a necessary part of a grid map from DRAM and storing it to the small BRAM buffer will also degrade the performance due to the repetitive small data transfers from DRAM to BRAM. This necessitates a BRAM storage for the entire map to minimize the memory access latency and the number of data transfers.

Storing the entire map on BRAM is, however, also infeasible, since it exhausts BRAM resources especially on small FPGA devices and increases a transfer overhead. The grid map resolution needs to be fine enough for the precise pose estimation ($r = 0.05$m in this paper), and contains tens of thousands of grid cells as a result. For instance, grid maps in Figure 3 contain 102,400 cells to cover 20.0×12.8m area.

The proposed CSM core limits the size of grid maps up to 320×320 cells or 16×16m, which is sufficiently large considering the typical indoor environment (e.g., office). From this limitation, the upper-bound of search space size becomes

16×16m (i.e., $2w_x, 2w_y \leq 320$ and $\widehat{w}_x, \widehat{w}_y \leq 40$). Only a region visible from a LiDAR is transferred to the core, which is reasonable considering the characteristics of rotating LiDAR sensors. Since the measurement range is upper-bounded, only a fraction of the map around the current sensor position is necessary for matching: the other remaining part distant from the sensor can be omitted. In the matching modules (vi)-(vii), scan points outside the boundary of trimmed grid maps are just ignored and not considered in the score evaluation.

Grid map values are also quantized to 6-bit (64 discrete values) due to the scarcity of BRAM slices: if 32-bit floating-point format is used, 65% of BRAM available in Pynq-Z2 is consumed for each map buffer, and the above design (Figure 6) becomes not synthesizable. The CSM core stores only the high-order 6-bits of retrieved 8-bit values to the buffers. As shown in the evaluation (Section IV), the algorithm performance is not severely affected by the quantization errors.

D. Parallelizing the Fine Matching

The matching modules (vi)-(vii) parallelize score evaluations by exploiting partially sequential access patterns found in CSM (Figure 7). Fine matching (Algorithm 1, lines 11-15) evaluates matching scores for w^2 solution candidates ranging from (n'_x, n'_y, n_θ) to $(n'_x + w - 1, n'_y + w - 1, n_\theta)$ using Equation 3. Observation of Equation 3 reveals that, for the k-th scan point, grid map values from $\mathcal{M}(i_k + n'_x, j_k + n'_y)$ to $\mathcal{M}(i_k + n'_x + w - 1, j_k + n'_y + w - 1)$ are accessed (Figure 7 (left)), to evaluate scores for the above candidates.

This motivates to develop the parallelized version of fine matching as shown in Algorithm 2, which firstly interchanges the loops over k and n_x, and then completely unrolls the innermost n_x-loop by setting the unrolling factor to w, so that the spatial locality in map accesses is exploited. It also parallelizes the loop over n_y. By using Algorithm 2, our fine matching module can process $2w$ consecutive grid map elements and compute $2w$ scores $\{s[\cdot,\cdot]\}$ in parallel, with only a small memory overhead. With the cyclic partitioning of $\mathcal{M}$ along x, y dimensions, the latency for fine matching is reduced from 235 to 15 μs ($w = 8, N = 360$), yielding the 15.74$\times$ performance improvement.

Algorithm 2 Parallelized Fine Matching

1: **for** $n_y = n'_y, n'_y + 2, \ldots, n'_y + w - 2$ **do**
2: $\quad \forall i \in [0, \ldots, w-1], j \in [0, 1]\ \ s[i,j] \leftarrow 0$
3: $\quad$ **for** $k = 1, \ldots, N$ **do**
4: $\quad\quad \forall i, j,\ s[i,j] \leftarrow s[i,j] + \mathcal{M}(i_k + n'_x + i, j_k + n_y + j)$
5: $\quad i^*, j^* \leftarrow \arg\max_{i,j} s[i,j]$
6: $\quad$ **if** $s[i^*, j^*] > s^*$ **then**
7: $\quad\quad s^* \leftarrow s[i^*, j^*], (n^*_x, n^*_y, n^*_\theta) \leftarrow (n'_x + i^*, n_y + j^*, n_\theta)$

E. Parallelizing the Coarse Matching

Our design also parallelizes coarse matching (Algorithm 1, lines 6-10), by reordering map elements and applying the similar optimizations as above. For the k-th scan point, grid map elements starting from $\mathcal{M}(i_k - w_x, j_k - w_y)$ to

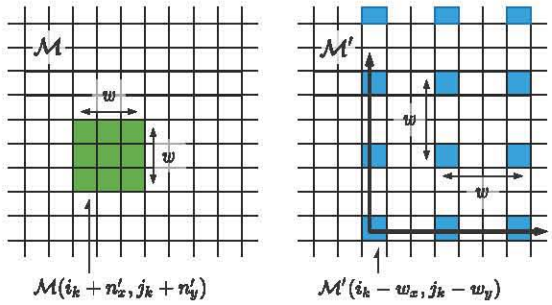

Fig. 7. Access patterns in fine (left) and coarse (right) matching ($w = 3$)

$\mathcal{M}(i_k - w_x + (\widehat{w}_x - 1)w, j_k - w_y + (\widehat{w}_y - 1)w)$ are accessed when iterating over n'_x and n'_y (i.e., coarse search space), creating strided access patterns with the stride of w (Figure 7 (right)). From this observation, parallelized coarse matching algorithm is obtained (Algorithm 3), which unrolls the loop n'_x, thereby allowing the parallel evaluation of scores (Equation 2). Note that the horizontal order of coarse map elements is rearranged as depicted in Figure 8 (left), to convert the stride accesses to non-stride sequential accesses (Figure 8 (right)). The unrolling factor of loop n'_x is set to eight, meaning that eight consecutive elements along x-dimension are accessed and eight scores are computed in parallel (line 5).

Algorithm 3 Parallelized Coarse Matching

1: **for** $n'_y = -w_y, -w_y + w, \ldots, -w_y + (\widehat{w}_y - 1)w$ **do**
2: $\quad$ **for** $n'_x = -w_x, -w_x + 8w, \ldots, -w_x + (\widehat{w}_x - 8)w$ **do**
3: $\quad\quad \forall i \in [0, \ldots, 7],\ s'[i] \leftarrow 0$
4: $\quad\quad$ **for** $k = 1, \ldots, N$ **do**
5: $\quad\quad\quad \forall i,\ s'[i] \leftarrow s'[i] + \mathcal{M}'(i_k - w_x + i \cdot w, j_k + n'_y)$
6: $\quad\quad$ **for** $i = 0, \ldots, 7$ **do**
7: $\quad\quad\quad$ **if** $s'[i] > s^*$ **then**
8: $\quad\quad\quad\quad$ Perform fine matching (Algorithm 2)

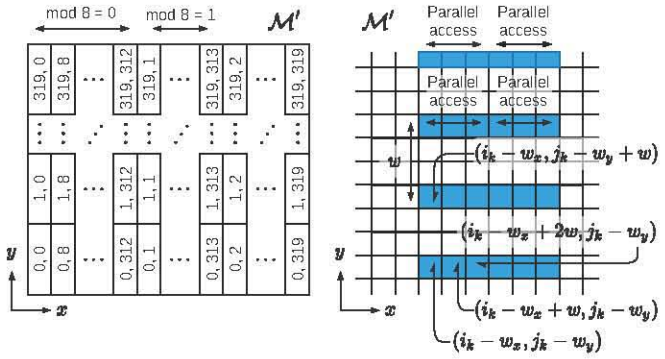

Fig. 8. Rearranging the layout of coarse map $\mathcal{M}'$ (left, $w = 8$) to parallelize the strided accesses and optimized access patterns (right, $w = 3$)

F. Reusing Previously Transferred Data

Using flag packets (Figure 5), the CSM core provides an option to skip data transfers, and reuses the previously transferred ones already stored on BRAM. This improves the

performance of one-to-many scan matching, i.e., matchings between a set of scans and a map $(\mathcal{M}, \{\mathcal{S}_1, \ldots, \mathcal{S}_K\})$, or between a set of grid maps and a scan $(\{\mathcal{M}_1, \ldots, \mathcal{M}_K\}, \mathcal{S})$. In the first case, a map $\mathcal{M}$ is transferred only once before the matching with $\mathcal{S}_1$, and then is reused for matchings with $\mathcal{S}_2, \ldots, \mathcal{S}_K$, eliminating $K-1$ unnecessary transfers of $\mathcal{M}$ and $K-1$ precomputations of $\mathcal{M}'$ in the module (iii).

G. Integration into SLAM

Our CSM core is easily integrated into LiDAR SLAM systems that use scan-to-map matching. As depicted in Figure 9, we offload the scan matching part in two major algorithms: PF-SLAM and graph-based SLAM. PF-SLAM (Figure 9 (left)) creates two threads to parallelize scan matching for particles: each thread independently uses one pair of CSM core and DMA controller to process half of the particles, which involves the matching between one scan and multiple grid maps individually owned by particles. Other tasks, e.g., updates of grid maps and particle resampling, are executed on the CPU.

Graph-based SLAM (Figure 9 (right)) is also multi-threaded and assigns one CSM core to each thread. The frontend thread performs scan matching between the latest scan and a grid map consisting of recent scans to update the current robot pose. The backend thread performs loop detections by attempting to match the recent scans against old submaps consisting of previously acquired scans, i.e., performs many-to-many scan matching. In the algorithmic aspect, loop detection is basically the same as the frontend scan matching but with the larger search space, and thus the CSM core is also applicable to loop detections without modifying any logic. The backend treats a matching attempt as a success, if a returned score s^* is greater than the predefined threshold s_T. The next section evaluates the performance of our core.

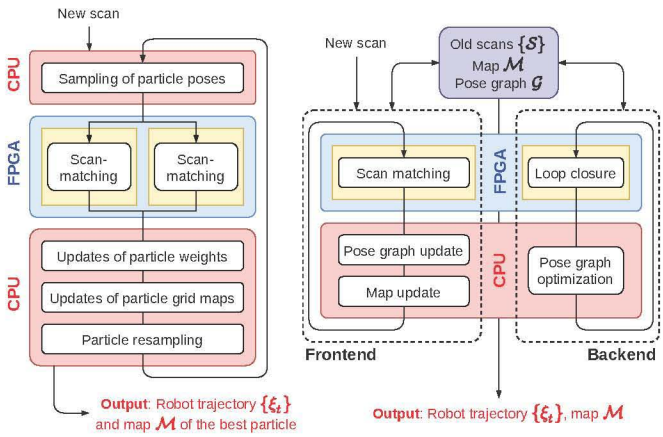

Fig. 9. Acceleration of the particle filter-based (left) and graph-based (right) LiDAR SLAM

IV. Implementation and Experimental Results

A. Details of the CSM Core Implementation

We used Xilinx Vivado HLS 2019.2 for developing the proposed core in C++, and Vivado 2019.2 to run synthesis and place-and-route steps. Pynq-Z2 development board (Figure 10 (left)) is chosen as a target device, to show that our design fits within the low-priced and resource-constrained FPGAs. Pynq-Z2 consists of Xilinx XC7Z020-1CLG400C FPGA fabric (equivalent to Artix-7), dual-core ARM Cortex-A9 CPU at 650MHz, and 512MB DDR3 DRAM. It runs Pynq Linux based on Ubuntu 18.04. The operation frequency of our design is set to 100 MHz.

B. Details of the SLAM Implementations

Our PF-SLAM and graph-SLAM systems were written in C++ from scratch without ROS (Robot Operating System). Their designs are inspired by the famous GMapping [18] and the representative graph-based methods (e.g., Karto SLAM [24] and Google Cartographer [1]), respectively. The graph-SLAM backend uses the g^{2}o library [25] for pose graph optimization. We compiled them using GCC 7.3.0 with `-O3` compiler flag. Note that the software scan matcher in PF-SLAM is parallelized using OpenMP for the fair performance comparison. In PF-SLAM, we used 16 particles throughout the experiments presented below. As mentioned in Section II-C, CSM estimates the robot position at $r = 5\text{cm}$ accuracy, which is same as the grid map resolution. After performing CSM, our SLAM system refines the pose estimate $\boldsymbol{\xi}^*$ at a subpixel accuracy using iterative scan matching methods [30] [32].

C. Real-world Datasets

We used three publicly available datasets from the Radish repository [33], recorded at Intel Research Lab (**Intel**, $28.5 \times 28.5\text{m}$), MIT CSAIL Building (**MIT**, $61 \times 46.5\text{m}$), and ACES Building (**ACES**, $56 \times 58\text{m}$). We also obtained a real-world dataset at the corridor in an office building using a Hokuyo URG-04LX-UG01 range finder, which has the maximum measurement range of 5.6m and the angular resolution of 0.36° (Figure 10 (right)). The wheeled robot was controlled remotely to move around the corridor twice for 1,226 seconds, and collected scans and wheel odometry data at around 500ms interval (2Hz). We also used Revo LDS dataset from [1], which includes scan data captured at around 200ms interval (5Hz) using the low-cost Revo LDS LiDAR.

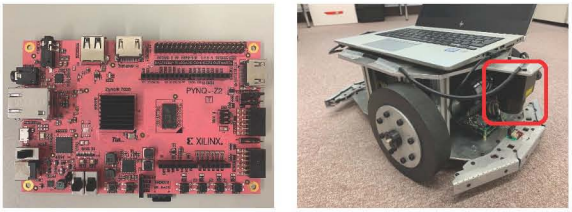

Fig. 10. Pynq-Z2 development board (left) and wheeled mobile robot equipped with Hokuyo URG-04LX-UG01 LiDAR sensor (red square) (right)

D. Resource Utilization and Runtime Memory Consumption

The FPGA resource utilization of our design (Figure 4) is summarized in Table I. The BRAM consumption is linear with the number of CSM cores implemented and also with the size of grid map buffers (Figure 6), thus our design is constrained

by the amount of BRAM available. Grid map buffers consume 68.6% (96 slices) of BRAMs, which would increase 5.3x without quantizing values from 32-bit to 6-bit.

TABLE I
FPGA RESOURCE UTILIZATION OF THE CSM CORE

	BRAM	DSP	FF	LUT
Used	111	24	20,121	21,026
Available	140	220	106,400	53,200
Utilization (%)	79.29	10.91	18.91	39.52

In graph-based SLAM, the software scan matcher stores coarse maps to the DRAM to avoid computations for the same input, which reduces the preprocessing cost when matching multiple scans to the same grid map (i.e., loop detections). In contrast, CSM core computes coarse maps inside the sliding window maximum module (Figure 6) and eliminates the necessity of DRAM storage for caching coarse maps. Using CSM core, the physical DRAM usage is reduced from 59.5 to 48.3 MiB (**ACES**), 101.7 to 76.6 MiB (**Intel**), and 60.1 to 39.5 MiB (**MIT-CSAIL**), respectively.

E. Execution Time Breakdown and Latency

Figures 11 and 12 show the execution time breakdowns of PF-SLAM and graph-based SLAM frontend. Both figures consider the data transfer overhead between PS-PL for fair comparisons. In PF-SLAM, CSM core speeds up the scan matching process by 13.67x, 14.09x, and 13.67x, reducing the total execution time by 4.34x, 4.67x, and 4.03x on **ACES**, **Intel**, and **MIT-CSAIL**, respectively, which demonstrates the effectiveness of CSM core on variety of datasets. Similarly, in graph-based SLAM, CSM core speeds up the frontend scan matching by 14.84x, 13.90x, and 13.85x, reducing the total execution time by 4.00x, 3.14x, and 3.09x on those datasets.

The above performance improvement is achieved by parallelizing the coarse-to-fine matching as described in Section III, and by offloading coarse map precomputations to the CSM core. The computational complexity of this precomputation is linear with the number of grid cells in a given map and has the measurable impact on the entire performance: in PF-SLAM and graph-based SLAM, this accounted up to 36.9% and 45.1% of the scan matching process.

Compared to PF-SLAM that maintains a set of particles and considers multiple hypotheses on robot trajectory and map, graph-based SLAM computes only the most plausible estimate, leading to the better performance. Figures 11 and 12 highlight the performance advantage of graph-based SLAM over PF-SLAM (e.g., 98.2ms and 416.2ms per frame in **Intel**). The backend loop detection in graph-based SLAM is accelerated by 16.18x (3760.1 to 75.4ms per successful detection), 17.23x (544.2 to 28.3ms) and 16.64x (577.1 to 37.0ms) on **ACES**, **Intel**, and **MIT-CSAIL** datasets, indicating that CSM core provides better performance improvements with a larger search space. CSM core offered comparable loop detection capability: it successfully detected 115, 491, and 141 loops, whereas the software version detected 42, 407, and 46 loops from those datasets. Note that the comparisons presented here are only indicative, since different grid maps and scans are used for loop detections on CPU and FPGA due to the timing differences.

In Revo LDS dataset, the latency for processing a single scan was 63.7 ± 39.4ms in graph-based SLAM, which is shorter than the scan period (i.e., 200ms), and hence the real-time performance is achieved for most of the runtime.

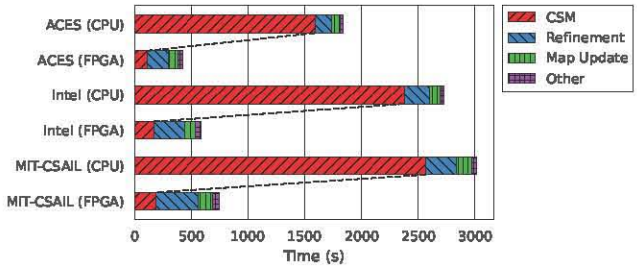

Fig. 11. Execution time breakdown of the PF-SLAM. It took 584.4s (416.2ms per frame) to process the **Intel** dataset with FPGA acceleration (4th row), which is 4.67x speedup compared to the CPU-only execution (3rd row).

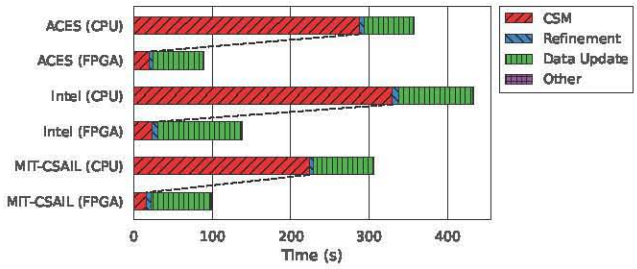

Fig. 12. Execution time breakdown of the graph-based SLAM frontend. It took 137.9s (98.2ms per frame) to process the **Intel** dataset with FPGA acceleration (4th row), which is 3.14x speedup compared to the CPU-only execution (3rd row).

F. Accuracy

Figures 1, 14 show the grid maps obtained from CSM cores to demonstrate the quality of outputs. Note that the grid map for Revo LDS dataset is created using only LiDAR scans and without odometry information. Figure 13 compare the robot trajectories obtained from CSM cores against the ones from software scan matchers and pure odometry (i.e., only using wheel encoders to estimate robot motions, red line). The significant overlap between two trajectories (CPU and FPGA) indicate that the output of CSM core preserves the accuracy of the software scan matchers, in spite of the quantization of grid map values, limitation of the buffer sizes, and rounding errors introduced by fixed-point arithmetic operations.

Tables II and III quantitatively evaluate the accuracy of output robot trajectories obtained from Radish datasets. Since ground-truth trajectories are not provided in Radish datasets, we used the accuracy measure proposed in [34] that does not require them. The accuracy is defined by the errors between the robot motion $\Delta\boldsymbol{\xi}_{i,j} = \boldsymbol{\xi}_j \ominus \boldsymbol{\xi}_i$ from time i to j, which is computed from the output trajectory $\{\boldsymbol{\xi}_0, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_T\}$, and the ground truth motions $\Delta\boldsymbol{\xi}^*_{i,j}$ obtained by manually matching the scan data $\mathcal{S}_i, \mathcal{S}_j$ at time i and j using the dedicated

software. Note that the inverse compounding operator $\ominus$ computes a relative pose between two given poses.

As shown in Tables II and III, CSM core achieved accuracy close to the software implementation and another similar work [12], where the authors of [12] propose an FPGA-based accelerator for the iterative scan matching and integrate it to PF-SLAM. Compared to [12], CSM core can handle backend loop detections with large initial errors as well as the frontend scan matching, showing its robustness and versatility, while at the same time providing faster computation time. Translational errors were less than the twice of the grid map resolution in **ACES** and **MIT-CSAIL** datasets, and were also comparable to that of the state-of-the-art method [1], especially in PF-SLAM. PF-SLAM presented better accuracy than graph-based SLAM, which is predictable, as PF-SLAM maintains multiple hypotheses about the current state (i.e., robot trajectory) and selects the most probable one, which yields robustness and improved accuracy at the cost of increased computational costs (Figures 11 and 12). Figure 14 (right) shows the grid map of the corridor: we can confirm that the graph-based SLAM successfully detected and closed a loop when a robot moved around the corridor.

TABLE II
COMPARISON OF THE TRAJECTORY ACCURACIES (PF-SLAM)

		CPU	FPGA
ACES	Trans (m)	0.082 ± 0.196	0.058 ± 0.067
	Rot (rad)	0.087 ± 0.322	0.087 ± 0.323
Intel	Trans (m)	0.132 ± 0.155	0.118 ± 0.130
	Rot (rad)	0.088 ± 0.287	0.087 ± 0.286
MIT	Trans (m)	0.040 ± 0.045	0.041 ± 0.046
	Rot (rad)	0.044 ± 0.286	0.043 ± 0.286

TABLE III
COMPARISON OF THE TRAJECTORY ACCURACIES (GRAPH-BASED SLAM)

		CPU	FPGA
ACES	Trans (m)	0.078 ± 0.108	0.092 ± 0.131
	Rot (rad)	0.217 ± 0.626	0.213 ± 0.617
Intel	Trans (m)	0.152 ± 0.227	0.146 ± 0.162
	Rot (rad)	0.174 ± 0.504	0.139 ± 0.418
MIT	Trans (m)	0.062 ± 0.109	0.064 ± 0.118
	Rot (rad)	0.160 ± 0.542	0.143 ± 0.474

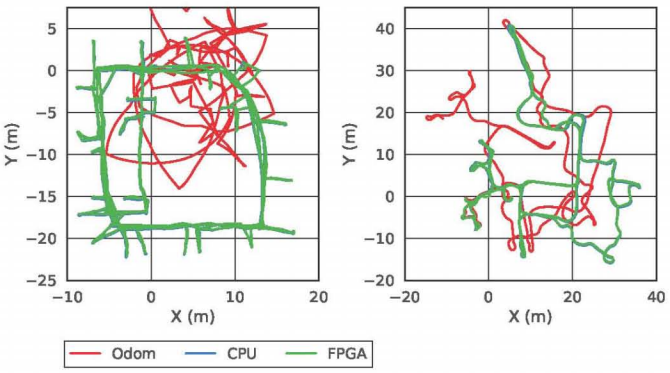

Fig. 13. Comparison of the trajectories (left: PF-SLAM, **Intel**, right: graph-based SLAM, **MIT-CSAIL**)

G. Power Consumption

We measured the power consumption of the entire Pynq-Z2 board using a wattmeter: the power consumption was 2.3-2.4W when running both PF-SLAM and graph-based SLAM systems with and without our proposed CSM cores. Note that the value reported above include the power consumption of CPU and other peripherals as well as that of CSM cores, meaning that the CSM core itself consumes less than 2.3-2.4W.

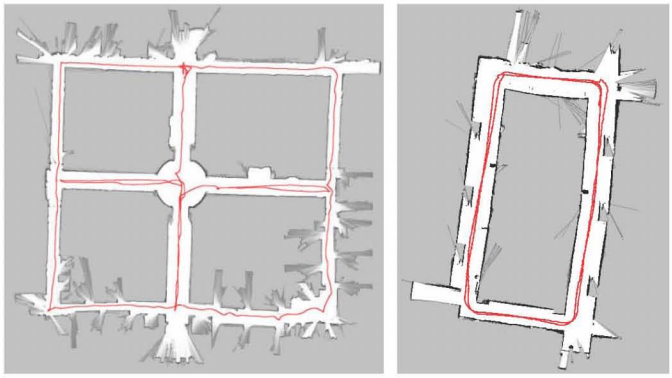

Fig. 14. Grid map and robot trajectory (red line) obtained using our proposed CSM core (left: graph-based SLAM and **ACES** dataset, right: graph-based SLAM and dataset captured at the corridor)

V. CONCLUSION

In this paper, we proposed an FPGA-based accelerator, which is applicable for a variety of 2D LiDAR SLAM methods. We focused on the scan matching part as it becomes the major bottleneck, and chose to implement Correlative Scan Matching (CSM) method considering the balance between algorithm simplicity and efficiency. We conducted several architectural and algorithmic optimizations such as data realignments and loop transformations, to fully exploit the inherent parallelism and minimize the resource utilization. Our design consisting of two CSM cores can be implemented on a low-cost FPGA with severe resource constraints (Pynq-Z2 board).

For evaluations, we integrated the proposed accelerator into two representative SLAM approaches, namely PF-SLAM and graph-based SLAM. We confirmed that CSM core improves the scan matching performance by up to 14.09x and 14.84x, and the overall performance by up to 4.67x and 4.00x in these SLAM methods while only consuming 2.3-2.4W. Loop detection in graph-based SLAM is accelerated by up to 17.23x. Results also highlighted the advantages of these methods: graph-based SLAM achieved faster computation time, whereas PF-SLAM provided more accurate trajectories and maps, and our unified design can easily switch between these methods. The error of trajectory estimate was around 10cm and 0.1rad in most cases, which was comparable to that of the software counterparts and even the state-of-the-art SLAM method.

ACKNOWLEDGMENTS

This work was supported by JST CREST Grant Number JPMJCR18K1, Japan.

REFERENCES

[1] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[2] E. Pedrosa, A. Pereira, and N. Lau, "Efficient Localization based on Scan Matching with a Continuous Likelihood Field," in *Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2017, pp. 61–66.

[3] D. Barnes, R. Weston, and I. Posner, "Masking by Moving: Learning Distraction-Free Radar Odometry from Pose Information," arXiv preprint arXiv:1909.03752, Sep. 2019.

[4] F. Nie, W. Zhang, Z. Yao, Y. Shi, F. Li, and Q. Huang, "LCPF: A Particle Filter Lidar SLAM System With Loop Detection and Correction," *IEEE Access*, vol. 8, pp. 20 401–20 412, Jan. 2020.

[5] C. Schulz and A. Zell, "Real-Time Graph-Based SLAM with Occupancy Normal Distributions Transforms," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3106–3111.

[6] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 431–437.

[7] Q. Gautier, A. Shearer, J. Matai, D. Richmond, P. Meng, and R. Kastner, "Real-time 3D Reconstruction for FPGAs: A Case Study for Evaluating the Performance, Area, and Programmability Trade-offs of the Altera OpenCL SDK," in *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, 2014, pp. 326–329.

[8] K. Boikos and C.-S. Bouganis, "Semi-Dense SLAM on an FPGA SoC," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[9] ——, "A High-Performance System-on-Chip Architecture for Direct Tracking for SLAM," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[10] Q. Gautier, A. Althoff, and R. Kastner, "FPGA Architectures for Real-time Dense SLAM," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2019, pp. 83–90.

[11] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eSLAM: An Energy-Efficient Accelerator for Real-Time ORB-SLAM on FPGA Platform," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[12] K. Sugiura and H. Matsutani, "An FPGA Acceleration and Optimization Techniques for 2D LiDAR SLAM Algorithm," *IEICE Transactions on Information and Systems*, vol. E104.D, no. 6, pp. 789–800, Jun. 2021.

[13] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-LOAM: Fast LiDAR Odometry And Mapping," arXiv preprint arXiv:2107.00822, Jul. 2021.

[14] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles," *IEEE Transactions on Intelligent Transportation Systems (TITS)*, vol. (Early Access), no. (Early Access), pp. 1–15, Mar. 2021.

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

[16] G. Grisetti, C. Stachniss, and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 667–672.

[17] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and Accurate SLAM with Rao-Blackwellized Particle Filters," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, Jan. 2007.

[18] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 32–46, Mar. 2007.

[19] B. D. Gouveia, D. Portugal, and L. Marques, "Speeding Up Rao-Blackwellized Particle Filter SLAM with a Multithreaded Architecture," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 1583–1588.

[20] B. Sileshi, J. Oliver, R. Toledo, J. Gonçalves, and P. Costa, "On the behaviour of low cost laser scanners in HW/SW particle filter SLAM applications," *Robotics and Autonomous Systems*, pp. 11–23, 2016.

[21] B. Sileshi, J. Oliver, and C.Ferrer, "Accelerating Particle Filter on FPGA," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 591–594.

[22] A. Krishna, A. V. Schaik, and C. S. Thakur, "FPGA Implementation of Particle Filters for Robotic Source Localization," *IEEE Access*, vol. (Early Access), no. (Early Access), p. (Early Access), Jul. 2021.

[23] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," *IEEE Transactions on Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, Dec. 2010.

[24] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, and R. Vincent, "Efficient Sparse Pose Adjustment for 2D Mapping," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 22–29.

[25] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A General Framework for Graph Optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3607–3613.

[26] D. lee, H. Kim, and H. Myung, "GPU-Based Real-Time RGB-D 3D SLAM," in *Proceedings of the International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Nov. 2012, pp. 46–48.

[27] A. Ratter, C. Sammut, and M. McGill, "GPU Accelerated Graph SLAM and Occupancy Voxel Based ICP for Encoder-Free Mobile Robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2013, pp. 540–547.

[28] A. Ratter and C. Sammut, "Fused 2D/3D Position Tracking for Robust SLAM on Mobile Robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS)*, Sep. 2015, pp. 1962–1969.

[29] E. Olson, "M3RSM: Many-to-Many Multi-Resolution Scan Matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5815–5821.

[30] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on Standardization in Mobile Robot Programming : The Carnegie Mellon Navigation (CARMEN) Toolkit," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 2436–2441.

[31] 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.

[32] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011, pp. 155–160.

[33] A. Howard and N. Roy, "The Robotics Data Set Repository (Radish)," http://radish.sourceforge.net/, 2003.

[34] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, Nov. 2009.