PAPER

An FPGA Acceleration and Optimization Techniques for 2D LiDAR SLAM Algorithm[*]

Keisuke SUGIURA[†a)], ***Nonmember and*** **Hiroki MATSUTANI**[†b)], ***Member***

SUMMARY An efficient hardware implementation for Simultaneous Localization and Mapping (SLAM) methods is of necessity for mobile autonomous robots with limited computational resources. In this paper, we propose a resource-efficient FPGA implementation for accelerating scan matching computations, which typically cause a major bottleneck in 2D LiDAR SLAM methods. Scan matching is a process of correcting a robot pose by aligning the latest LiDAR measurements with an occupancy grid map, which encodes the information about the surrounding environment. We exploit an inherent parallelism in the Rao-Blackwellized Particle Filter (RBPF) based algorithm to perform scan matching computations for multiple particles in parallel. In the proposed design, several techniques are employed to reduce the resource utilization and to achieve the maximum throughput. Experimental results using the benchmark datasets show that the scan matching is accelerated by 5.31–8.75× and the overall throughput is improved by 3.72–5.10× without seriously degrading the quality of the final outputs. Furthermore, our proposed IP core requires only 44% of the total resources available in the TUL Pynq-Z2 FPGA board, thus facilitating the realization of SLAM applications on indoor mobile robots.

key words: *SLAM, GMapping, SoC, FPGA*

1. Introduction

Simultaneous localization and mapping (SLAM) technology plays an indispensable role in autonomous robots, such as autonomous driving cars and cleaning robots, and has been a major research topic in robotics over the last two decades. In order to operate in a previously unknown environment, autonomous robots need to estimate its vehicle pose by matching the sensor observation against the current map, while updating the current map based on the current pose and sensor observation. Due to this structure of mutual dependence between the robot pose and map, localization and mapping cannot be handled independently from each other. SLAM algorithms aim to solve these two problems simultaneously.

The Bayes filter-based approach has been widely applied to the SLAM problem. The variation of Bayes filters including Extended Kalman Filter (EKF) [1] and particle filter are utilized in the process. FastSLAM [2], [3] and GMapping [4] are the most popular methods among particle filter-based approaches and are proven to work well in the literature [5]. GMapping is the grid-based LiDAR SLAM based on Rao-Blackwellized Particle Filter (RBPF). It takes odometry information and measurements from Light Detection and Ranging (LiDAR) sensors as input and generates a sequence of robot poses (trajectory) and an occupancy grid map, which discretize the surrounding environment into equal-sized square cells.

Although SLAM is the key component and basis for autonomous mobile robots, its high computational requirement emerges as a major problem when using SLAM in these robots. SLAM requires high-end CPUs and sometimes even GPUs to handle massive computations [6]–[8]. However, there is a situation where these CPUs and GPUs cannot be mounted because of limited power budgets, costs, and physical constraints (size or weight). Consequently, there exists a strong demand for hardware accelerators to execute SLAM algorithms on such robots. Hardware offloading brings certain benefits, e.g. performance improvement without additional power consumption.

Particle filter is performed using a set of particles, where each particle carries a single hypothesis of the current state (i.e. robot trajectory and map). Fortunately, operations on these particles are independent of each other; therefore such an algorithm is suitable for FPGAs with parallel processing capability. In this paper, an FPGA-based accelerator for GMapping is proposed, by making use of the inherent parallel properties in the algorithm. Experimental results using benchmark datasets demonstrate that the FPGA accelerator is a feasible solution for improving the throughput without significantly degrading the accuracy.

The rest of this paper is organized as follows. Section 2 presents a brief description for GMapping and its theoretical foundation. In Sect. 3, related works for hardware acceleration of RBPF-based SLAM algorithms are reviewed. In Sect. 4, the FPGA accelerator for GMapping is proposed, and its architectural and algorithmic optimizations are described. Section 5 illustrates the implementation details. Evaluation results in terms of throughput, accuracy, resource utilization, and power consumption are shown in Sect. 6. Section 7 concludes this paper.

2. Preliminaries

2.1 Rao-Blackwellized Particle Filter

Rao-Blackwellized Particle Filter (RBPF), an extension of particle filter, is a powerful tool for solving the so-called

Manuscript received September 1, 2020.
Manuscript revised January 12, 2021.
Manuscript publicized March 4, 2021.

[†]The authors are with Graduate School of Science and Technology, Keio University, Yokohama-shi, 223–8522 Japan.

[*]This work was supported, in part, by JST CREST Grant Number JPMJCR18K1, Japan.

a) E-mail: sugiura@arc.ics.keio.ac.jp
b) E-mail: matutani@arc.ics.keio.ac.jp

DOI: 10.1587/transinf.2020EDP7174

full SLAM problem [4], [9], [10]. Full SLAM is expressed in the form of the following posterior distribution (1) over the state variables consisting of the robot map $m = \{m_i\}$ and robot trajectory $x_{1:t} = \{x_1, \ldots, x_t\}$, conditioned on the sequence of sensor observations $z_{1:t} = \{z_1, \ldots, z_t\}$ and robot controls $u_{1:t} = \{u_1, \ldots, u_t\}$.

$$p(m, x_{1:t} | z_{1:t}, u_{1:t}) \tag{1}$$

In particle filters, the above posterior (1) is represented by a swarm of particles. A major drawback is that the number of particles required to sufficiently approximate the posterior grows exponentially with the dimension of the state space. In the context of SLAM, state variables (i.e. robot pose and map) usually reside in a very high-dimensional space (up to tens of thousands of dimensions). Therefore, the original particle filter cannot be applied since it would require an enormous amount of particles. To address this, the posterior (1) is decomposed into two terms as shown in Eq. (2) using the chain rule, which correspond to the trajectory distribution, and the map posterior conditioned on the robot trajectory, respectively [11].

$$p(m, x_{1:t} | z_{1:t}, u_{1:t}) = p(x_{1:t} | z_{1:t}, u_{1:t}) p(m | x_{1:t}, z_{1:t}) \tag{2}$$

In RBPF, only the robot trajectory $x_{1:t}$ is estimated by a particle filter; that is, the set of particles tries to approximate the posterior (**target**) distribution P_t over the trajectory

$$\begin{aligned} P_t &\equiv p(x_{1:t} | z_{1:t}, u_{1:t}) \\ &= \eta \, p(z_t | x_{1:t}, z_{1:t-1}) p(x_t | x_{t-1}, u_t) P_{t-1} \\ &\simeq \eta \, p(z_t | x_t, m) p(x_t | x_{t-1}, u_t) P_{t-1} \end{aligned} \tag{3}$$

The map m is computed deterministically as a function of the trajectory estimate $x^*_{1:t}$ and the observations $z_{1:t}$. In this case, the map distribution $p(m | x_{1:t}, z_{1:t})$ can be viewed as a Gaussian with zero variance, where all probability mass is concentrated at the particular point $x_{1:t} = x^*_{1:t}$. Hence, the approximation in Eq. (3) holds, as below:

$$\begin{aligned} & p(z_t | x_{1:t}, z_{1:t-1}) \\ & \quad = \int p(z_t | x_t, m') p(m' | x_{1:t-1}, z_{1:t-1}) dm' \\ & \quad \simeq p(z_t | x_t, m) \end{aligned} \tag{4}$$

Each particle individually carries the map as well as trajectory, since the map depends on the estimated trajectory, which differs for each particle. This factorization yields a significant reduction of the number of particles (i.e. computational cost) because particles are drawn from the relatively low-dimensional space P_t containing robot trajectory only. The kth particle at time t, and the particle set at time t are denoted as $Y^{[k]}_t = \{x^{[k]}_t, m^{[k]}, w^{[k]}_t\}$ and $\mathcal{S}_t = \{Y^{[1]}_t, \ldots, Y^{[M]}_t\}$ respectively, where M is the number of particles. RBPF follows the general Sampling Importance Resampling (SIR) algorithm and is outlined by the following four steps.

In the first *sampling* step, a new particle pose $x^{[k]}_t$ is sampled from the Gaussian motion model $p(x^{[k]}_t | x^{[k]}_{t-1}, u_t)$, which represents the motion uncertainty usually caused by sensor errors, wheel slippages or surface irregularities. At this point, the set of the particle trajectories $\{x^{[k]}_{1:t}\}$ reflects the prior (**proposal**) distribution Q_t given in Eq. (5).

$$Q_t = p(x_t | x_{t-1}, u_t) P_{t-1} \tag{5}$$

Then, in *map update* step, the scan data z_t is inserted into each particle map $m^{[k]}$ based on the current particle pose $x^{[k]}_t$, which will be described in detail later. After that, in *weight update* step, an importance weight associated to each particle $w^{[k]}_t$ is updated based on the ratio between the target P_t and proposal Q_t:

$$w^{[k]}_t = w^{[k]}_{t-1} \frac{P_t}{Q_t} \simeq \eta \, w^{[k]}_{t-1} p(z_t | m^{[k]}, x^{[k]}_t) \tag{6}$$

In (6), $p(z_t | m^{[k]}, x^{[k]}_t)$ is the observation likelihood which models the underlying generating process of an observation given the map $m^{[k]}$ and current robot pose $x^{[k]}_t$. In other words, it represents the consistency of observed data z_t with a map and pose. Lastly, in *resampling* step, a new generation of particles $\mathcal{S}_t$ is obtained by resampling the particles (allowing duplication) with probability proportional to the importance weights. Particles with small weights are removed and those with large weights are likely to dominate the entire population. Particles $\{x^{[k]}_{1:t}\}$ now distribute according to the desired posterior distribution P_t, which appears in Eq. (2). Resampling process is crucial for transforming the particle distribution from prior (proposal) to posterior (target).

2.2 GMapping

GMapping is classified as the RBPF-SLAM algorithm and is commonly used among the robotics community. It periodically retrieves the latest robot control u_t and scan data $z_t = \{z^i_t\}$ captured from a LiDAR sensor. It then builds a planar occupancy grid map m, in which each grid cell contains a probability that the cell is occupied by an object. A single observation $z^i_t = [r^i_t, \theta^i_t]^\top$ is comprised of distance r^i_t and angle θ^i_t with respect to the sensor.

GMapping employs two strategies to reduce the computational burden: improved proposal and adaptive resampling. In the *sampling* step, a new particle pose $x^{[k]}_t$ is drawn from the altered distribution (7) instead of the raw odometry motion model $p(x_t | x_{t-1}, u_t)$.

$$p(x_t | m, x_{t-1}, z_t, u_t) = \frac{p(z_t | m, x_t) p(x_t | x_{t-1}, u_t)}{\displaystyle\int p(z_t | m, x) p(x | x_{t-1}, u_t) dx} \tag{7}$$

The above distribution (7) also takes into account the latest observation z_t and is more peaked than the ordinary motion model, thereby providing a highly accurate pose x_t [4]. To perform a sampling based on Eq. (7), the robot pose $x'^{[k]}_t$ is initially sampled from the motion model $p(x_t | x_{t-1}, u_t)$ and then is refined so that the current scan z_t and map $m^{[k]}$ maximally overlap each other. This alignment is called *scan matching*, and involves the maximization of the likelihood function formalized as below.

$$x_t^{[k]} = \arg\max_x p(x|m^{[k]}, z_t, x_t'^{[k]}) \tag{8}$$

It leads the particles to be located in a more meaningful area with higher observation likelihood, thus reducing the number of particles and improving algorithmic efficiency. The proposal now takes the following form

$$Q_t' = p(x_t|m, x_{t-1}, z_t, u_t)P_{t-1}. \tag{9}$$

The importance weight is then computed as follows

$$\begin{aligned} w_t^{[k]} &= w_{t-1}^{[k]} \frac{P_t}{Q_t'} \\ &= \eta\, w_{t-1}^{[k]} \frac{p(z_t|x_t^{[k]}, m^{[k]})p(x_t^{[k]}|x_{t-1}^{[k]}, u_{t-1})}{p(x_t^{[k]}|m^{[k]}, x_{t-1}^{[k]}, z_t, u_t)} \\ &= \eta\, w_{t-1}^{[k]} \int p(z_t|m^{[k]}, x)p(x|x_{t-1}^{[k]}, u_t)dx. \end{aligned} \tag{10}$$

Since the observation likelihood has a much smaller variance than the motion model, the integral above may be evaluated around the maximum of the likelihood, $x_t^{[k]}$, which is already obtained as a result of scan matching. Consequently, the weight computation (10) is further simplified to Eq. (11).

$$w_t^{[k]} \simeq \eta\, w_{t-1}^{[k]} p(z_t|m^{[k]}, x_t^{[k]}) \tag{11}$$

Resampling is only performed when the effective sample size in Eq. (12) falls below the threshold value M_{th}.

$$M_{\text{eff}} = \frac{1}{\sum_k \left(w_t^{[k]}\right)^2} \tag{12}$$

M_{eff} can be interpreted as the accuracy of the proposal. It reaches its maximum value M when all weights are identical ($w_t^{[k]} = M^{-1}$), that is, the proposal distribution fully reflects the target distribution. An excessive variance of the importance weights incurs a small M_{eff}. Especially when M_{eff} is large, resampling is unnecessary since the current particle set is assumed to represent the target distribution effectively. The adaptive resampling technique enables to retain the diversity of hypotheses and thus mitigates the risk of the particles around the correct state being removed, also known as particle deprivation (depletion).

Algorithm 1 summarizes the overall algorithm of GMapping, where the symbol $\oplus$ denotes the composition operator [12] and ε is the zero-mean Gaussian noise.

The function AddScan(m, x_t, z_t) incorporates the scan data z_t into the map m using the robot position x_t. It transforms each scan $z_t^i = \left[r_t^i, \theta_t^i\right]^\top$ from the sensor coordinate to the map coordinate and computes the hit point (also referred to as the beam endpoint) p_t^i. Then it determines the *hit* grid cell that contains p_t^i and *missed* grid cells that lie on the straight line connecting p_t^i and x_t using Bresenham's algorithm. Binary Bayes filter is applied to these grid cells and their occupancy probabilities are incrementally updated. The probability values associated with *missed* cells are lowered since they are less likely to be obstructed (laser rays just went through these cells), and opposite for the *hit* cell.

Algorithm 1 GMapping Algorithm

```
function GMapping()
    $t \leftarrow 1, \quad \mathcal{S}_0 \leftarrow \varnothing$
    for $k = 1, \ldots, M$ do                                   ▷ Initialize particle set
        $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \left\{x_0, m_0, M^{-1}\right\}$
                        ▷ Set initial pose, empty grid map, and initial weight
    while $\{u_t, z_t\}$ exists do                               ▷ Consume sensor data
        $\mathcal{S}_t \leftarrow$ Process($\mathcal{S}_{t-1}, z_t, u_t$), $\quad t \leftarrow t + 1$
    $k^* \leftarrow \arg\max_k \left\{w_t^{[k]}\right\}$
                        ▷ Choose the best particle with largest importance
    return $x_{1:t}^{[k^*]}, m^{[k^*]}$
                        ▷ Return the most plausible trajectory and map

function Process($\mathcal{S}_{t-1}, z_t, u_t$)
    $\mathcal{S}_t = \varnothing$                                ▷ Initialize new particle set
    for each $Y_{t-1}^{[k]} \in \mathcal{S}_{t-1}$ do
        $x' \leftarrow x_{t-1}^{[k]} \oplus u_t + \varepsilon$                ▷ Initial guess
        $x_t^{[k]} \leftarrow \arg\max_x p(x|m^{[k]}, z_t, x')$          ▷ Scan matching
        $m^{[k]} \leftarrow$ AddScan($m^{[k]}, x_t^{[k]}, z_t$)          ▷ Update map
        $w_t^{[k]} \leftarrow \eta\, w_{t-1}^{[k]} \int p(x|x_{t-1}^{[k]}, u_t)p(z_t|x, m^{[k]})dx$
                                                               ▷ Update weight
        $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \left\{x_{1:t}^{[k]}, m^{[k]}, w_t^{[k]}\right\}$     ▷ Add to new particle set

    $M_{\text{eff}} = \left[\sum_k \left(w_t^{[k]}\right)^2\right]^{-1}$
                                          ▷ Compute effective sample size
    if $M_{\text{eff}} < M_{\text{th}}$ then
        $\mathcal{S}_t \leftarrow$ Resample($\mathcal{S}_t$)               ▷ Resample if necessary

    return $\mathcal{S}_t$
```

3. Related Work

There are several works on accelerating RBPF-based SLAM methods for embedded platforms by exploiting their parallel nature [13]–[17]. Abouzahir *et al.* quantitatively analyzed execution times of SLAM algorithms under varying parameter settings and concluded that FastSLAM 2.0 is preferable for the low-cost embedded systems in terms of the real-time performance and consistency of output results [18]. Their implementation of the Monocular FastSLAM 2.0 targeting CPU-FPGA heterogeneous architectures outperformed those run on high-end CPU or GPU and demonstrated the feasibility of FPGA as an accelerator in the domain of SLAM. FastSLAM 2.0 is also a variant of the RBPF-based method as GMapping [3]. The primary difference is that FastSLAM 2.0 builds a feature-based map, consisting of the features of landmarks recognized by robots, while GMapping constructs a grid-based map.

Both map representations are widely used; however, the former requires the feature extraction and detection from sensor inputs, i.e., prior knowledge about the environment structure. The main advantage of the grid-based map is its flexibility, meaning that it can represent arbitrary objects and thus no assumption of the environment is needed [19]. Also, occupancy state at any location is easily obtainable owing to its dense data structure, making it a convenient format for other tasks such as navigation and motion planning, which are based on pathfinding algorithms. From the aspect of the scan matching using LiDAR data, the matching between a scan and a grid map (often referred to as *scan-to-map*) generally produces accurate and robust alignments than the matching between two scans (*scan-to-scan*) [6].

The major drawback of the grid-based map is that it demands a large amount of memory in exchange for its dense representation [19]. This problem is even more critical in RBPF-based SLAM, because each particle keeps its individual map, which means that the memory consumption grows quadratically to the map size and also proportionally to the number of particles. However, several techniques are proposed to mitigate the problem by sharing a part of grid map among multiple particles [10], [20], [21], exploiting the implicit redundancy in the particle maps. That is, multiple identical copies of a single particle map are created during a *resampling* process, but only tiny fractions of them are modified and large parts remain unchanged. In our software implementation, the map sharing technique similar to [20] is applied to reduce the memory consumption and to increase the maximum number of particles. These techniques are effective especially when the RBPF-SLAM is being run on a resource-limited platform. From the above considerations, the grid-based approach is focused in this paper. To the best of our knowledge, this is the first work that presents FPGA design for grid-based RBPF-SLAM.

Gouveia *et al.* proposed a multithreaded version of GMapping using the OpenMP library, and high map precision was gained by increasing the number of particles without sacrificing the latency [22]. Li *et al.* also examined an acceleration of GMapping leveraging several parallel processing libraries [23]. The above-mentioned works focus on GMapping acceleration from the software aspects. In this paper, on the other hand, we investigate the FPGA implementation of GMapping for the first time and propose optimization methods to achieve resource efficiency and high-performance.

4. Design Optimization

In Sect. 4.1, we first provide a reason for choosing the scan matching part as a target of the hardware acceleration. We then thoroughly describe the algorithm for scan matching in Sect. 4.2 and three optimization techniques adopted in the hardware implementation in Sects. 4.3–4.5.

4.1 Parallelization of Scan Matching

As described in Sect. 2.2, the algorithm is divided into five main parts: *initial guess*, *scan matching*, *map udpate*, *weight update*, and *resampling*. Its notable feature is that all the operations except resampling can be performed simultaneously for multiple particles. Scan matching is the process of superimposing a scan on a grid map, i.e. it tries to find the most suitable alignment so that a map and a scan projected onto a map maximally overlap each other. It inevitably becomes time-consuming and computationally intensive [24], since a large number of calculations (especially coordinate transformations) and random accesses to the map are required. Performance evaluations in Sect. 6 reveal that scan matching accounts for up to 90 % of the total execution time, clearly posing a major bottleneck. Scan matching is the most reasonable candidate for hardware acceleration in terms of the expected performance gain. In this paper, as illustrated in Fig. 1, scan matching is executed in parallel on an FPGA device and other necessary computations are handled on the CPU side, thus utilizing the heterogeneous SoC architecture.

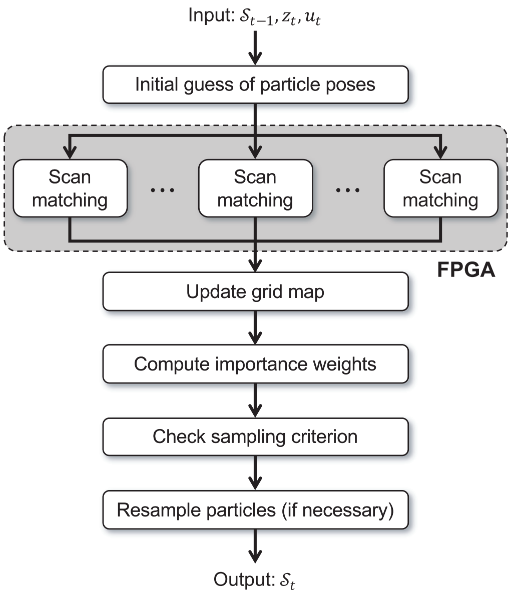

Input: $\mathcal{S}_{t-1}, z_t, u_t$
Initial guess of particle poses
Scan matching
…
Scan matching
…
Scan matching
FPGA
Update grid map
Compute importance weights
Check sampling criterion
Resample particles (if necessary)
Output: $\mathcal{S}_t$

Fig. 1 Parallelization of scan matching

4.2 Greedy Endpoint Matching Algorithm

The software implementation used in this paper is based on the open-source package provided by OpenSLAM [25]. In the OpenSLAM GMapping package, a metaheuristic hill-climbing based algorithm called Greedy Endpoint Matching [26] is executed during the scan matching process. It is worth noting that more sophisticated algorithms like branch-and-bound based method [6] and correlation-based method [27] can be applied for scan matching. Although the hill-climbing method has a weakness that its performance is negatively affected by the poor initial estimates and is susceptible to local optima [27], a comparison of the scan matching algorithms' performance is outside the scope of this paper.

The hill-climbing algorithm corrects a particle pose x_t by aligning a scan data $z_t = \left\{z_t^i\right\}$ with a map m. More concretely, a particle pose x_t that maximizes a matching score $s(x_t, m, z_t)$ is continually explored until a convergence is reached. The matching score is regarded as the observation likelihood $p(x_t|m, z_t, x_t')$ as mentioned in Sect. 2, where x_t' denotes an initial estimate of a particle pose. In each iteration, the algorithm chooses an axial direction that most improves the score, and then the particle pose is updated by a small step along that direction. The update step, which is analogous to a learning rate in gradient descent optimization, is halved if the score is not improved and no feasible direction is found, and the algorithm ends if a convergence criterion is met (i.e. the update step becomes sufficiently small). The score $s(x_t, m, z_t)$ is calculated according to the following equation.

$$s(x_t, m, z_t) = \sum_i \exp\left\{ -\frac{\left(d_t^i\right)^2}{2\sigma^2} \right\} = \sum_i u(d_t^i), \tag{13}$$

where σ is the predefined standard deviation and summand $u(d_t^i)$ is the score for ith measurement z_t^i. The d_t^i denotes the distance between the ith scan point p_t^i (described in Sect. 2) and its closest obstacle registered in the map m. A smaller value of d implies a small misalignment between the observation z_t and the map m. Scan point p_t^i of the ith observation $z_t^i = [r_t^i, \theta_t^i]^\top$ is computed by the coordinate transformation from the sensor frame to the map frame under the current pose $x_t = [\xi_t^x, \xi_t^y, \xi_t^\theta]^\top$ as follows.

$$p_t^i = \begin{bmatrix} \xi_t^x + r_t^i \cdot \cos(\xi_t^\theta + \theta_t^i) \\ \xi_t^y + r_t^i \cdot \sin(\xi_t^\theta + \theta_t^i) \end{bmatrix} \in \mathbb{R}^2 \tag{14}$$

The naive yet stable algorithm to find the minimum distance d_t^i is summarized in Algorithm 2. $\gamma(x^{\mathrm{m}}) : \mathbb{R}^2 \to \mathbb{Z}^2$ is a function that converts the position in the map frame $x^{\mathrm{m}} = \left[\xi_x^{\mathrm{m}}, \xi_y^{\mathrm{m}}\right]^\top$ to the corresponding grid cell index. It is formulated as

$$\gamma(x^{\mathrm{m}}) = \begin{bmatrix} \lfloor (\xi_x^{\mathrm{m}} - o_x)/\Delta \rfloor \\ \lfloor (\xi_y^{\mathrm{m}} - o_y)/\Delta \rfloor \end{bmatrix} \in \mathbb{Z}^2, \tag{15}$$

where Δ is a map resolution (grid cell size) and $[o_x, o_y]^\top$ denotes the position of the map origin (the position that corresponds to the grid cell with a minimum index $(0, 0)$), respectively. $\gamma^{-1}(C_x, C_y)$ is the inverse of γ, written as

$$\gamma^{-1}(C_x, C_y) = \begin{bmatrix} o_x + C_x\Delta \\ o_y + C_y\Delta \end{bmatrix} \in \mathbb{R}^2. \tag{16}$$

Algorithm 2 first calculates the scan point p_t^i and its closest grid cell C^{H} for each scan z_t^i. It then calculates $\widetilde{p}_t^i$ and C^{M} in the same way. $\widetilde{p}_t^i$ is the point that is closer to the sensor by δ than the scan point p_t^i. The cell C^{M} is therefore presumed to be unoccupied and missed by the beam (i.e. C^{M} should belong to the set of *missed* grid cells, because the laser beam passed through the cell C^{M}). Figure 2 (left) shows an example of the positional relationship between p_t^i and $\widetilde{p}_t^i$.

After that, it attempts to establish the matching between the observation z_t^i and the map m. It utilizes a square searching window of $(2K + 1) \times (2K + 1)$ cells, centered at the C^{H} (see Fig. 2 (left)). In our implementation, the radius K is currently set to 1, yielding the 3×3 square searching window. Every grid cell covered by the window is considered a candidate for containing the beam endpoint p_t^i. That is, p_t^i might not reside in the C^{H} but in proximity to the C^{H} because of the accumulated error in x_t or the perturbation in measurement z_t^i. The searching window is to allow these errors and to consider the case where p_t^i does not exactly correspond to C^{H}.

Algorithm 2 Calculation of d_t^i

```
function FindMinimumDistance($x_t, m, z_t^i$)
    $p_t^i \leftarrow \begin{bmatrix} \xi_t^x + r_t^i \cos(\xi_t^\theta + \theta_t^i) \\ \xi_t^y + r_t^i \sin(\xi_t^\theta + \theta_t^i) \end{bmatrix}$
                                                        ▹ Compute scan point
    $\widetilde{p}_t^i \leftarrow \begin{bmatrix} \xi_t^x + (r_t^i - \delta) \cos(\xi_t^\theta + \theta_t^i) \\ \xi_t^y + (r_t^i - \delta) \sin(\xi_t^\theta + \theta_t^i) \end{bmatrix}$
                                        ▹ Compute point that seems unoccupied
    $C^{\mathrm{H}} \leftarrow \gamma(p_t^i)$                          ▹ Compute hit cell index
    $C^{\mathrm{M}} \leftarrow \gamma(\widetilde{p}_t^i)$                     ▹ Compute missed cell index
    $d^* \leftarrow \infty$                                ▹ Initialize minimum distance
    for $k_x = -K, \ldots, K$ do
        for $k_y = -K, \ldots, K$ do
                                        ▹ For each cell in searching window
            $\widetilde{C}^{\mathrm{H}} \leftarrow (C_x^{\mathrm{H}} + k_x, C_y^{\mathrm{H}} + k_y),\ p^{\mathrm{H}} \leftarrow m(\widetilde{C}^{\mathrm{H}})$
            $\widetilde{C}^{\mathrm{M}} \leftarrow (C_x^{\mathrm{M}} + k_x, C_y^{\mathrm{M}} + k_y),\ p^{\mathrm{M}} \leftarrow m(\widetilde{C}^{\mathrm{M}})$
                               ▹ Check occupancy probabilities of candidate cells
            if $p^{\mathrm{H}} > T$ and $p^{\mathrm{M}} < T$ then
                $p' \leftarrow \gamma^{-1}(\widetilde{C}^{\mathrm{H}}),\ d^* \leftarrow \min(d^*, |p_t^i - p'|)$
                                  ▹ Update minimum distance if criteria met
    return $d^*$
```

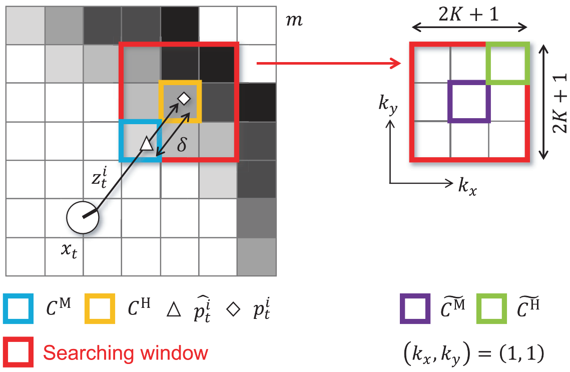


Fig. 2 Scan point and its surroundings

Each cell in the searching window and its associated occupancy probability are denoted as $\widetilde{C}^{\mathrm{H}}$ and p^{H}, respectively. The index of $\widetilde{C}^{\mathrm{H}}$ is given by adding a relative offset (k_x, k_y) to C^{H} (refer to Fig. 2 (right)). The same applies to p^{M} and $\widetilde{C}^{\mathrm{M}}$. For each cell $\widetilde{C}^{\mathrm{H}}$, it is tested whether two values p^{H} and p^{M} are within the desired ranges: $(T, 1]$ and $[0, T)$. These criteria are derived from the fact that $\widetilde{C}^{\mathrm{H}}$ and $\widetilde{C}^{\mathrm{M}}$ should be *hit* and *missed* cell. If $\widetilde{C}^{\mathrm{H}}$ satisfies these criteria, $\widetilde{C}^{\mathrm{H}}$ is the appropriate matching candidate and is expected to accommodate the scan point p_t^i, meaning that p_t^i actually resides in $\widetilde{C}^{\mathrm{H}}$ and not C^{H}. The distance between p_t^i and p' is then calculated, where p_t^i is the scan point obtained from the current pose x_t using Eq. (14), and p' is its corresponding point found on the map m, respectively. The minimum distance is selected for d_t^i if multiple grid cells satisfy the criteria. Checking the value of p^{M}, which is expected to be lower than the p^{H}, effectively avoids the false matching and hence contributes to the robustness.

The optimizations to realize the resource-efficient implementation are threefold: (a) map compression, (b) efficient access to map data, and (c) simplified score calculation.

4.3 Map Compression

The map resolution Δ is preferred to be set to a smaller value, e.g. 0.01 m or 0.05 m, since it directly affects the accuracy of the output map. More importantly, the RBPF-based approach requires map hypotheses to be maintained individually on each particle. The amount of memory needed to store the map increases approximately to the square of the map size, inversely to the square of the map resolution Δ, and also proportional to the number of particles M. Typically, it ranges in the order of hundreds of megabytes, especially when a considerable number of particles are used to deal with a mapping in a relatively large environment. On an FPGA platform with limited hardware resources, the amount of FPGA on-chip memory (BRAM) is not enough for even storing one single map, and thus frequent data transfer between the BRAM and an on-board DRAM will be required. In addition, transferring such amount of data imposes a massive overhead, which potentially outweighs the advantage of hardware acceleration. As a result, an effective way of reducing the map size should be devised.

Considering the physical principle of a LiDAR sensor, it is immediately apparent that only a fraction of the mapped area is observable from a sensor at any iteration. This indicates that the *local* map covering only the surrounding of the robot can be utilized during the scan matching process instead of the entire map, a significant part of which is eventually not used. Local map $\widetilde{m}$ is essentially a cropped version of the original map m. Local map for kth particle $\widetilde{m}^{[k]}$ is constructed by clipping an area of the predetermined size of $2W \times 2W$ grid cells from the map $m^{[k]}$, centering on the grid cell (C_x, C_y) corresponding to the current pose $x_t^{[k]}$ (see Fig. 3).

$$\widetilde{m}^{[k]} = \left\{ m^{[k]}(C_x + k_x, C_y + k_y) | k_x, k_y \in [-W, W) \right\} \quad (17)$$

This amounts to the approximation of proposal distribution $p(x_t | m, x_{t-1}, z_t, u_t)$ by substituting the map m with the local map $\widetilde{m}$ [10]. In the current implementation, Δ and W are set to 0.05 m and 128, respectively, making a local map 12.8 m square. W should be selected so that almost every scan point fits inside the local map; otherwise, the accuracy of scan matching is seriously lost. The scan points that are out of the local map are not taken into account in the score evaluation (Eq. (13)) and the algorithm greatly suffers from the resulting inaccurate score. In an environment densely occupied with obstacles, smaller W is applicable, since the distance to the nearest obstacle (obtained as a scan data from a laser scanner) tends to become relatively shorter. Use of local maps clearly reduces both hardware amount and data transfer latency. As a side benefit, each map can be viewed as a fixed-size 2D array from the FPGA side, thus facilitating data retrieval and processing. On the software, the map is implemented as a variable-sized array and is dynamically expanded when a robot enters previously unexplored areas, whereas the size of the local map remains unchanged.

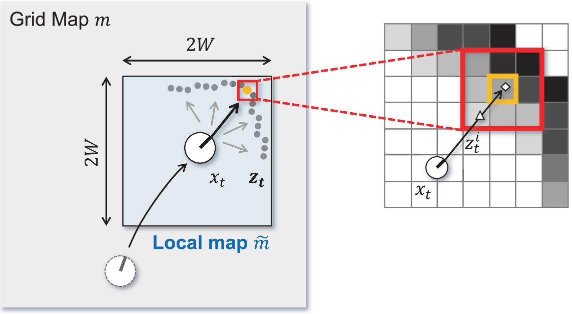
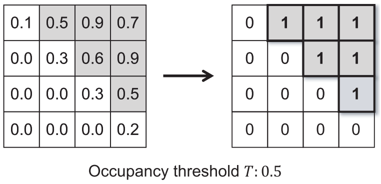


Fig. 3 Entire grid map and local map



0.1	0.5	0.9	0.7
0.0	0.3	0.6	0.9
0.0	0.0	0.3	0.5
0.0	0.0	0.0	0.2

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

Fig. 4 Local map binarization

An occupancy value is stored in a double-precision floating-point format in the software implementation. According to Algorithm 2, however, one can find that the floating-point representation is completely redundant since the value is only used for the comparison against the occupancy threshold T; the value itself is not of interest. For this reason, occupancy values can be quantized into 1-bit values by performing this comparison before being fed to the FPGA scan matcher core (Fig. 4). This binarization reduces resource usage by up to 64× with no accuracy loss and it finally becomes feasible to store local maps for multiple particles on BRAM blocks for parallel processing. Also, time-consuming DRAM accesses from inside of an FPGA are fully eliminated and the data transfer overhead is substantially reduced. Overall latency is also reduced in the way that a comparison between two floating-point numbers (appears in Line 11 in Algorithm 2) is turned into a simple bit operation.

4.4 Efficient Access to Map Data

As mentioned in Sect. 4.2, the searching window has the size of 3×3 grid cells. Under this setting, one can observe that the single execution of Algorithm 2 results in eighteen consecutive accesses to the grid cells in map m; nine for the *hit* cells $\widetilde{C}^{\mathrm{H}}$ and the other nine for the *missed* cells $\widetilde{C}^{\mathrm{M}}$. Minimizing the latency for the map data acquisitions (i.e. BRAM accesses) is crucial because it resides in the innermost part of the scan matching algorithm and thus it directly affects the entire performance of the IP core.

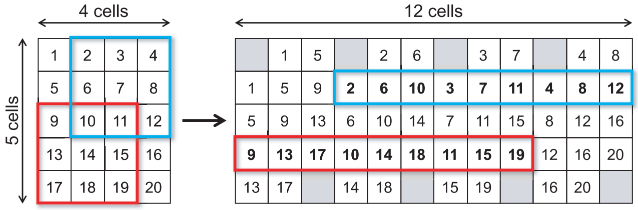
An example of the typical access pattern that occurs when sweeping a single searching window (consisting of nine elements) is shown in Fig. 5 (left). In this case, in order to obtain all nine elements in a single cycle, the map data (2D array) needs to be completely partitioned along both dimensions, thereby eating up valuable memory resources. In our FPGA scan matcher, however, the algorithm does not follow the above access pattern; instead, it accesses the data along a horizontal direction (with the vertical position being fixed) as depicted in Fig. 5 (right). Apparently, the amount of memory to store the map increases by 3×, since the map now needs to contain duplicate elements to achieve this dedicated access pattern. The primary advantage of the modification of the data layout is that the algorithm can query a searching window within a single clock cycle by partitioning the map along a horizontal axis only, without the necessity of full partitioning. Avoiding the unnecessary partitioning is effective for reducing the resource usage. Despite of the 3× increase of the memory footprint caused by allowing redundancy, it is still possible to keep multiple grid maps on BRAMs by combining the map binarization and cropping technique presented in Sect. 4.3. In this way, the minimum latency for the map data accesses is achieved, mitigating the negative effects on the resource utilization.

4.5 Simplified Score Calculation

According to Algorithm 2, d' is essentially the distance $d' = |p_t^i - p'|$ between the two grid cells C^{H} and $\widetilde{C}^{\mathrm{H}}$, which correspond to the scan point p_t^i and its actual point p' on the map m as described in Sect. 4.3, respectively. Inspecting the following Eq. (18) reveals that d can be computed from only the offsets k_x, k_y, and map resolution Δ by approximating p_t^i with $\gamma^{-1}(C^{\mathrm{H}})$; hence the absolute positions p_t^i, p' are unneeded.

$$\begin{aligned} d' &= \left|p_t^i - p'\right| \simeq \left|\gamma^{-1}(C^H) - \gamma^{-1}(\widetilde{C}^H)\right| \\ &= \sqrt{(C_x^H - \widetilde{C}_x^H)^2 + (C_y^H - \widetilde{C}_y^H)^2}\Delta \\ &= \sqrt{k_x^2 + k_y^2}\Delta \end{aligned} \tag{18}$$

It turns out that d' and $u(d') = \exp(-d'^2/2\sigma^2)$ are discrete functions of relative offsets $k_x, k_y \in [-K, K]$. Note that $u(d')$ is a scan matching score for a single observation that appears in Eq. (13). A lookup table of size $(2K+1)^2$ that contains the Gaussian $u(d')$ of every possible distance d' (i.e. every possible combinations of offsets k_x, k_y) can be computed beforehand. This lookup table can be fully partitioned and mapped as registers, since it consists of only nine elements when $K = 1$. This precomputation enables the effective evaluation of the score $s(x, m, z_t)$ since the computation of the Gaussian function in Eq. (13) is replaced by the single query to the lookup table entry.



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

	1	5		2	6		3	7		4	8
1	5	9	**2**	**6**	**10**	**3**	**7**	**11**	**4**	**8**	**12**
5	9	13	6	10	14	7	11	15	8	12	16
9	**13**	**17**	**10**	**14**	**18**	**11**	**15**	**19**	12	16	20
13	17		14	18		15	19		16	20	

Fig. 5 Layout of map data on BRAM

5. Implementation

We implemented a scan matcher IP core that performs the aforementioned Greedy Endpoint Matching algorithm in parallel using Xilinx Vivado HLS v2019.2 toolchain. We chose Pynq-Z2 development board [28] as a target device (Table 1), which is equipped with a programmable logic and a dual-core embedded processor, to demonstrate that the proposed core can be implemented in devices with severe resource constraints. The clock frequency of the IP core is set to 100 MHz.

Figure 6 depicts a brief overview of the board-level implementation. The Zynq processing system (PS) executes our software implementation of GMapping algorithm (described in Sect. 2) except the scan matching part, which is offloaded to the programmable logic (PL) portion. The PS passes the input data by communicating with the DMA controller to initiate the scan matcher IP core. The DMA controller automatically creates fixed-sized AXI4-Stream packets containing the input data on the DRAM and delivers them to the IP core. It also receives the AXI4-Stream packets returned from the IP core and writes the extracted result data to the specified address range of the DRAM.

The IP core takes the following inputs from the PS: (1) initial guess of the N particle poses $\{x_t'^{[k]}\}$, (2) N local maps $\{\widetilde{m}^{[k]}\}$, (3) the latest sensor measurements $z_t = \{z_t^i\}$, and (4) additional parameters, where N is a parallelization degree. (4) includes the relative position of the local map $\widetilde{m}^{[k]}$ with respect to the entire map $m^{[k]}$. The IP core then sends back

Table 1 Specifications of Pynq-Z2 board

OS	Pynq Linux (based on Ubuntu 18.04)
CPU	ARM Cortex-A9 @ 650MHz × 2
FPGA	Xilinx Zynq XC7Z020-1CLG400C (Artix-7)
DRAM	512MB (DDR3)

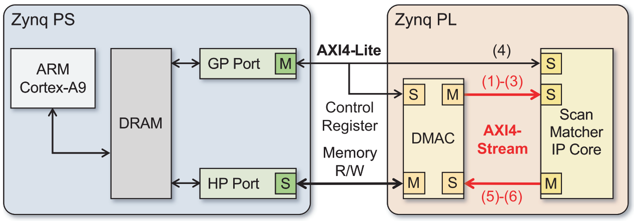


Fig. 6 Board level implementation

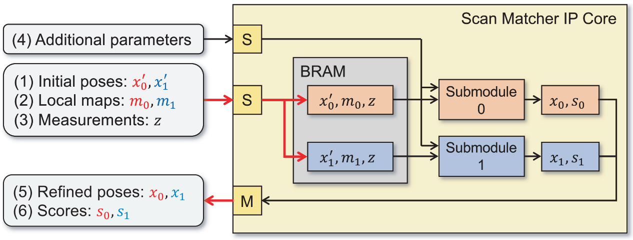


Fig. 7 Design of scan matcher IP core

(5) N refined particle poses $\{x_t^{[k]}\}$ and (6) final score values $\{s(x_t^{[k]}, m^{[k]}, z_t)\}$ associated to N particles to the PS; the latter can be used for weight computation. To complete the scan matching process for all particles, the IP core should be repetitively invoked for M/N times, where M is the total number of particles used. The input data that is shared among all particles (i.e. (3) and (4)) is transferred only once at the beginning of the scan matching phase. The DMA controller makes use of a high-performance port (HP Port) on the board and also adopts AXI4-Stream protocol for high-speed transmission of most of the input (1)–(3) and output (5)–(6). The other necessary parameters (4) are transferred via AXI4-Lite interface. At the beginning of the software implementation, the bitstream (binary image) of the IP core design is dynamically loaded to the PL using Linux kernel FPGA manager.

Figure 7 illustrates the block diagram of the proposed scan matcher core. The top module consists of two submodules, each of which computes the refined pose $x_t^{[k]}$ and the score value $s(x_t^{[k]}, m^{[k]}, z_t)$ for a single particle based on the Greedy Endpoint Matching algorithm, given the initial pose $x_t'^{[k]}$ and the local map $\widetilde{m}^{[k]}$. As a result, the IP core performs the scan matching for two particles at the same time, resulting in a parallelization degree of $N = 2$. Throughout our implementation, all the decimal numbers are represented by 32-bit fixed-point format with 16-bit signed integer and 16-bit fractional parts. These bitwidths are determined to preserve the adequate precision for values such as the linear and angular component of particle poses; however, the search for the optimal fixed-point number expression depends on a given application (or a surrounding environment) and is beyond the scope of this paper.

It is worth mentioning that, in the software implementation of the scan matching, the particle pose is repeatedly updated until it satisfies the convergence condition (i.e. update step of the particle pose is below the preset threshold, or the number of iterations exceeds the maximum). Conversely, in our IP core, the number of the optimization iterations is fixed (e.g. 25) in order to equalize the computational loads (latency) of all particles and realize the parallel execution. It is one of the (4) additional parameters as noted above and thus can be set from the processing system before invoking the IP core. We set this to 25 in all evaluations conducted in Sect. 6. Accordingly, the IP core maintains constant latency cycles as long as the number of particles is kept. Although this limitation typically causes the undesirable accuracy loss of the results, we observed that in most cases, the number of iterations is less than 25-30 and the average is around 10–15. Also, we did not see a significant degradation in terms of accuracy as shown in Sect. 6.

6. Evaluations

In this section, the proposed scan matcher IP core is evaluated in terms of algorithm latency, accuracy, FPGA resource utilization, and power consumption in comparison with the software implementation.

6.1 Experimental Setup

As a baseline, the entire GMapping algorithm is executed only with a CPU (ARM Cortex-A9 processor), which is denoted as **CPU**M (CPU, M particles) in this experiment. Then, the algorithm is executed with the CPU in cooperation with our IP core; that is, the CPU executes the software implementation of GMapping except the scan matching part, which is handled by our IP core. We refer to this experimental setting as **FPGA**M (FPGA, M particles). The software is developed in C++ and compiled using GCC 7.3.0 with `-O3` compiler flag to fully optimize the executable code.

The subset of publicly available Radish dataset [29], namely Intel Research Lab (**Intel**, 28.5m × 28.5m), ACES Buliding (**ACES**, 56m × 58m), and MIT CSAIL Building (**MIT-CSAIL**, 61m × 46.5m) is used for the benchmarking purpose. We chose these three datasets since they capture relatively small environments in which we expect our system to be run. The ground truth information is unavailable in these datasets; they only contain the sequence of sensor observations and odometry robot poses, making quantitative comparisons difficult. To measure the accuracy of output results (robot trajectories), we adopt the following performance metric proposed in [12].

$$\varepsilon_{t-1,t} = (x_t \ominus x_{t-1}) \ominus \delta^*_{t-1,t} \tag{19}$$

$$\varepsilon_{\text{trans}} = \frac{1}{T} \sum_t \|\text{trans}(\varepsilon_{t-1,t})\| \tag{20}$$

$$\varepsilon_{\text{rot}} = \frac{1}{T} \sum_t |\text{rot}(\varepsilon_{t-1,t})| \tag{21}$$

$$\sigma^2_{\text{trans}} = \frac{1}{T} \sum_t \left(\|\text{trans}(\varepsilon_{t-1,t})\| - \varepsilon_{\text{trans}}\right)^2 \tag{22}$$

$$\sigma^2_{\text{rot}} = \frac{1}{T} \sum_t \left(|\text{rot}(\varepsilon_{t-1,t})| - \varepsilon_{\text{rot}}\right)^2, \tag{23}$$

where $\ominus$ denotes the inverse composition operator, i.e. $x \ominus y$ represents the relative transformation between two poses x and y. Two helper functions trans(x) and rot(x) split a given pose $x = [\xi_x, \xi_y, \xi_\theta]^\top$ into two translational components (ξ_x, ξ_y) and an angular component (ξ_θ). $\|\cdot\|$ is a norm function ($\sqrt{\xi_x^2 + \xi_y^2}$) and $|\cdot|$ is an absolute value ($|\xi_\theta|$). The above metric computes the average and the standard deviation of discrepancies between two relative poses $x_t \ominus x_{t-1}$

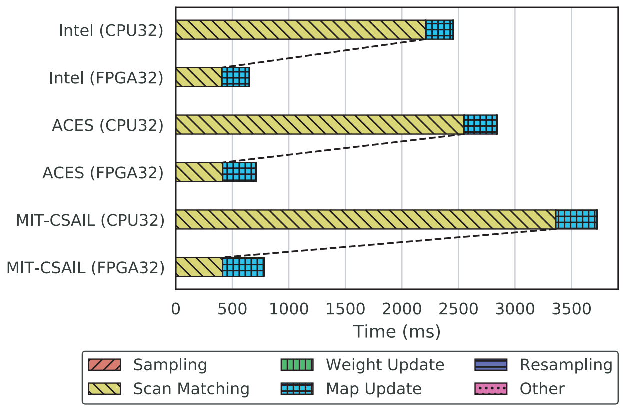


Fig. 8 Comparison of latency ($M = 32$)

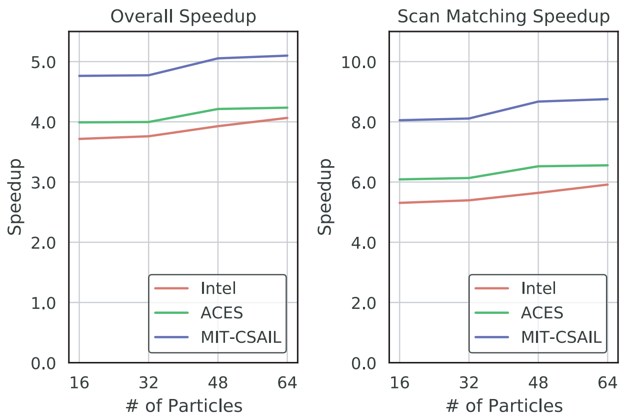


Fig. 9 Relationship between number of particles and speedup

and $\delta^*_{t-1,t}$; the former is the relative pose between temporally adjacent poses x_t and x_{t-1}, both of which are obtained from the trajectory result $x_{1:T}$. The latter is the ground truth relation extracted by manually matching the sensor observations (available at [30]). We also use the above metric to evaluate the difference (closeness) between the trajectories obtained from CPUM and FPGAM to confirm that our scan matcher IP core achieves competitive accuracy compared to the software implementation. We just substitute the $\delta^*_{t-1,t}$ in Eq. (19) with the relative pose $\widehat{x}_t \ominus \widehat{x}_{t-1}$, where $x_{1:T}$ and $\widehat{x}_{1:T}$ denote the trajectories from CPUM and FPGAM, respectively.

Table 2 Comparison of accuracy ($M = 32$)

	CPU32	FPGA32
ACES		
translational (m)	0.0558 ± 0.0649	0.125 ± 0.490
rotational (rad)	0.0851 ± 0.319	0.0852 ± 0.319
Intel		
translational (m)	0.115 ± 0.129	0.117 ± 0.130
rotational (rad)	0.0860 ± 0.284	0.0859 ± 0.284
MIT-CSAIL		
translational (m)	0.0483 ± 0.0764	0.0505 ± 0.0795
rotational (rad)	0.0970 ± 0.387	0.0984 ± 0.387

6.2 Algorithm Latency

Figure 8 shows the breakdown of the latency for a single iteration of the GMapping algorithm under two experimental configurations (CPU32 and FPGA32). The results presented here are the average of 5 executions. Note that the CPU-FPGA data transfer overhead is included in the scan matching latency for a fair comparison. We observed that most of the execution time is dominated by scan matching and map update processes; other processes contribute a negligible amount to the latency. The overall latency is effectively reduced up to ×4.77 (MIT-CSAIL) as a result of offloading the costly scan matching computations to the FPGA. For instance, in the Intel dataset, scan matching process accounts for 90.0 % of the total runtime in CPU32, representing a major bottleneck, while it accounts for 62.8 % in FPGA32. Though we adopted the high-performance streaming protocol, the data transfer still accounts for a large proportion of the scan matching latency. We attribute this to the memory-mapped I/O used to access the DMA controller registers or to handle input/output data. This indicates that if the overhead for memory-mapped I/O between PS and PL is minimized, the speedup ratio can be improved, though the software overhead for accessing/binarizing grid maps, and initializing/manipulating IP cores through memory-mapped I/O should also be reduced.

The relationship between the number of particles M and the speedup ratio is plotted in Fig. 9. Our hardware implementation achieves the approximately constant speedup but with slight increase (6.09–6.56× for ACES, 5.31–5.92× for Intel, and 8.05–8.75× for MIT-CSAIL, see Fig. 9 (left)) under the varying number of particles, thus demonstrating the scalability of our proposed system. The best speedup effect is obtained in the MIT-CSAIL dataset, in which the longest time is spent for scan matching computations among three datasets in the software implementation, while the latency of scan matching in our IP core remains constant regardless of the dataset used (see Sect. 5). We observed the same behavior in the overall speedup (3.99–4.24× for ACES, 3.72–4.07× for Intel, and 4.76–5.10× for MIT-CSAIL, see Fig. 9 (right)). Note that the slight increase of the speedup noticeable in three datasets comes from the slight performance degradation in the software implementation; we speculate that the consecutive accesses to the grid maps for a relatively large number of particles leads the increased cache miss rates in CPU. In MIT-CSAIL dataset, the scan matching latency for a single particle is 104.18 ms and 113.06 ms when $M = 16$ and $M = 64$, respectively, which means the increase of latency by 8.5 %.

6.3 Algorithm Accuracy

The accuracy of the output trajectories is measured based on the metric proposed in [12]. Table 2 compares the translational error $\varepsilon_{\text{trans}} \pm \sigma_{\text{trans}}$ and the rotational error $\varepsilon_{\text{rot}} \pm \sigma_{\text{rot}}$ obtained from FPGA32 against CPU32. This result presents the favorable performance of the FPGA32 except for the translational error in ACES dataset, which is due to its en-

Table 3 Difference in output trajectories ($M = 32$)

ACES	
translational (m)	0.0712 ± 0.127
rotational (rad)	0.00765 ± 0.00723
Intel	
translational (m)	0.0505 ± 0.0705
rotational (rad)	0.0134 ± 0.0266
MIT-CSAIL	
translational (m)	0.0495 ± 0.0641
rotational (rad)	0.0106 ± 0.0273

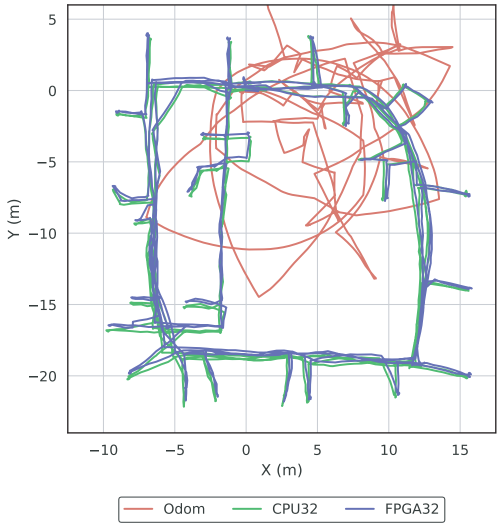


Fig. 10 Trajectories obtained from Intel Research Lab dataset

vironmental characteristics. ACES dataset mainly consists of long straight corridors, which makes the results of the scan matching (i.e. refined poses) unreliable; that is, the positional uncertainty in the longitudinal direction of the corridor tends to become large. FPGA32 is more likely to suffer from the occurrence of the unreliable scan matching than CPU32, since it uses the fixed-point representation for decimal values in the scan matching process, which introduces the propagation and accumulation of rounding errors in addition to the quickly accumulating positional errors.

Table 3 shows the difference (closeness) between the trajectories obtained from CPU32 and FPGA32, which is computed by slightly modifying Eq. (19) as explained in Sect. 6.1. Considering the map resolution ($\Delta = 0.05$m) and the angular resolution of the laser scanner (0.5°, 1.0°), it is obvious that the difference between two output trajectories is sufficiently small. The translational difference did not surpass 0.1 m in all datasets, which is equivalent to only two grid cells in a row. Despite the twofold increase of the translational error in ACES dataset (Table 2), we confirm that the relative error (0.0712 m) is within an acceptable range.

Figure 10 shows the robot trajectories obtained from CPU32 and FPGA32. The figure also shows the pure odometry trajectory, denoted as **Odom**. The considerable overlap between CPU32 and FPGA32 implies that the accuracy is not severely affected by introducing local maps as a part of map compression technique (Sect. 4.3). The scan points (obstacles) outside the local map are ignored in the score evaluations (Eq. (13)), which causes erroneous scan matching results especially when local maps are too small. In FPGA32, the computation based on fixed-point expressions introduces rounding errors, which would serve as a primary source of precision loss. FPGA32 is also affected by the limitation of the number of algorithm iterations (Sect. 5), by which the robot pose is not fully optimized and hence the cumulative error grows rapidly. Contrary to these concerns, FPGA32 still generates the topologically correct map and the underlying geometric relationship is maintained. In addition, the distortion and imprecision caused by these factors seem subtle, which is the satisfying outcome.

Table 4 FPGA resource utilization of scan matcher IP core (post place-and-route)

	BRAM	DSP	FF	LUT
Used	61	32	18,887	23,254
Available	140	220	106,400	53,200
Utilization (%)	43.6	14.6	17.8	43.7

6.4 FPGA Resource Utilization

Table 4 shows the FPGA resource utilization of our implementation, designed for Xilinx Zynq XC7Z020-1CLG400C assuming 100 MHz operating frequency. On-chip BRAMs are mostly consumed for the storage of local maps to execute the scan matching for multiple particles simultaneously, which implies that the BRAM consumption increases almost linearly proportional to the degree of parallelization. In our current design, the scan matching is parallelized for two particles, and the BRAM usage is still less than 50 % due to the map compression technique as described in Sect. 4.3. Especially, the extreme quantization of the occupancy value contributes to the resource reduction. The design uses certain amount of the LUT slices since mathematical operations (coordinate transformations) are frequently performed on the core. Though the achievable speedup is constrained by the total amount of BRAM and LUT resources present on a device, results in Table 4 suggest that other parts of the algorithm (i.e. importance weight calculation and initial pose guess) can be mapped onto the hardware. There is also enough room to increase the parallelization degree (e.g. 4) to achieve the further performance improvement.

6.5 Power Consumption

We used an ordinary watt-hour meter to measure the power consumption of the entire Pynq-Z2 board. Our board-level implementation (FPGA32) consumed 2.9 W of power, which is as same as the software-only implementation (CPU32). We emphasize that FPGA32 outperforms CPU32 in terms of the total execution time (3.76× shorter) when using Intel dataset as shown in Fig. 9.

7. Conclusions

The hardware optimization of SLAM methods is of crucial importance for deploying SLAM applications to autonomous mobile robots with severe limitations in power delivery and available resources. In this work, we proposed a lightweight FPGA-based design dedicated to accelerating the scan matching process in the 2D LiDAR SLAM method called GMapping by exploiting the parallel structure inherent in the algorithm. The resource usage and the overhead associated with the data transfers are effectively reduced by applying the map compression technique, which is the combination of map binarization and introduction of local maps. The map data is stored with the acceptable level of redundancy to enable the efficient data accesses thereby minimizing the latency. Also, the precomputed lookup table is employed to eliminate the expensive mathematical computations. Experiments based on benchmark datasets demonstrated that our hardware scan matcher avoids the loss of accuracy and offers satisfactory throughput to that of the software implementation. The proposed core achieved 5.31–8.75× scan matching speedup and 3.72–5.10× overall speedup. As far as we know, this is the first work that focuses on the hardware acceleration of the grid-based RBPF-SLAM.

References

[1] M.W.M.G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A Solution to the Simultaneous Localisation and Map Building (SLAM) Problem," IEEE Trans. Robot. Autom., vol.17, no.3, pp.229–241, June 2001.

[2] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," Proc. AAAI National Conference on Artificial Intelligence, pp.593–598, 2002.

[3] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," Proc. International Joint Conference on Artificial Intelligence (IJCAI), pp.1151–1156, June 2003.

[4] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," IEEE Trans. Robot., vol.23, no.1, pp.32–46, March 2007.

[5] J.M. Santos, D. Portugal, and R.P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," Proc. IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp.1–6, Oct. 2013.

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

[7] L. Nardi, B. Bodin, M.Z. Zia, J. Mawer, A. Nisbet, P.H.J. Kelly, A.J. Davison, M. Luján, M.F.P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," Proc. IEEE International Conference on Robotics and Automation (ICRA), pp.5783–5790, May 2015.

[8] Q. Gautier, A. Althoff, and R. Kastner, "FPGA Architectures for Real-time Dense SLAM," Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp.83–90, July 2019.

[9] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, 2005.

[10] 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.

[11] A. Doucet, N. de Freitas, K.P. Murphy, and S.J. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks," Proc. Conference on Uncertainty in Artificial Intelligence (UAI), pp.176–183, 2000.

[12] R. Kuemmerle, 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.

[13] K. Par and O. Tosun, "Parallelization of particle filter based localization and map matching algorithms on multicore/manycore architectures," Proc. IEEE Intelligent Vehicles Symposium (IV), pp.820–826, June 2011.

[14] M. Llofriu, F. Andrade, F. Benavides, A. Weitzenfeld, and G. Tejera, "An embedded particle filter SLAM implementation using an affordable platform," Proc. International Conference on Advanced Robotics (ICAR), pp.1–6, Nov. 2013.

[15] D. Portugal, B.D. Gouveia, and L. Marques, "A Distributed and Multithreaded SLAM Architecture for Robotic Clusters and Wireless Sensor Networks," Studies in Computational Intelligence, pp.121–141, May 2015.

[16] M. Abouzahir, A. Elouardi, S. Bouaziz, R. Latif, and A. Tajer, "Large-scale monocular FastSLAM2.0 acceleration on an embedded heterogeneous architecture," EURASIP Journal on Advances in Signal Processing, pp.1–20, 2016.

[17] 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, vol.80, pp.11–23, 2016.

[18] M. Abouzahir, A. Elouardi, R. Latif, and S. Bouaziz, "Embedding SLAM algorithms: Has it come of age?," Robotics and Autonomous Systems, pp.14–26, 2018.

[19] K.M. Wurm, C. Stachniss, and G. Grisetti, "Bridging the gap between feature- and grid-based SLAM," vol.58, no.2, pp.140–148, Feb. 2010.

[20] C. Schröter, H.J. Böhme, and H.M. Gross, "Memory-Efficient Gridmaps in Rao-Blackwellized Particle Filters for SLAM using Sonar Range Sensors," Proc. European Conference on Mobile Robots (EMCR), pp.138–143, Sept. 2007.

[21] H. Jo, H.M. Cho, S. Jo, and E. Kim, "Efficient Grid-Based Rao-Blackwellized Particle Filter SLAM With Interparticle Map Sharing," IEEE/ASME Trans. Mechatronics, vol.23, no.2, pp.714–724, April 2018.

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

[23] Q. Li, T. Rauschenbach, A. Wenzel, and F. Mueller, "EMB-SLAM: An Embedded Efficient Implementation of Rao-Blackwellized Particle Filter Based SLAM," Proc. International Conference on Control, Robotics and Cybernetics (CRC), pp.88–93, Sept. 2018.

[24] E. Olson, "M3RSM: Many-to-Many Multi-Resolution Scan Matching," Proc. IEEE International Conference on Robotics and Automation (ICRA), pp.5815–5821, 2015.

[25] C. Stachniss, U. Frese, and G. Grisetti, "OpenSLAM.org." https://openslam-org.github.io/, 2007.

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

[27] E.B. Olson, "Real-Time Correlative Scan Matching," Proc. IEEE

International Conference on Robotics and Automation (ICRA), pp.4387–4393, 2009.

[28] “TUL.” https://www.tul.com.tw/ProductsPYNQ-Z2.html.

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

[30] A. Kleiner, B. Steder, C. Dornhege, C. Stachniss, G. Grisetti, M. Ruhnke, R. Kümmerle, and W. Burgard, “SLAM Benchmarking Home.” http://ais.informatik.uni-freiburg.de/slamevaluation/, 2009.

Keisuke Sugiura received the BE degree from Keio University in 2020. He is currently a master course student in Keio University.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently an associate professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.