IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 January 2024, accepted 23 January 2024, date of publication 1 February 2024, date of current version 7 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3360883

RESEARCH ARTICLE

2D Particle Filter Accelerator for Mobile Robot Indoor Localization and Pose Estimation

OMER TARIQ AND DONGSOO HAN
School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Yuseong, Daejeon 34141, South Korea
Corresponding author: Omer Tariq (omertariq@kaist.ac.kr)

This work was supported by the Challengeable Future Defense Technology Research and Development Program through the Agency For Defense Development (ADD) funded by the Defense Acquisition Program Administration (DAPA) in 2023 under Grant 912906601.

ABSTRACT Particle filtering is a reliable Monte Carlo algorithm for estimating the state of a system in modeling non-linear, non-gaussian elements for estimation and tracking applications in various fields, including robotics, navigation, and computer vision. However, particle filtering can be computationally expensive, particularly in high-dimensional state spaces, and can be a bottleneck for real-time applications due to high memory consumption. This paper proposes a particle filter accelerator that employs a cellular automata-based pseudo-random number generator and an improved systematic resampler based on the Vose Alias method. The particles are distributed across several sub-filters, performing concurrent resampling and importance weights computations. The proposed accelerator leveraged the inherent parallelism and pipelining stages of FPGAs to perform the resampling stage in a parallel fashion, significantly enhancing the particle convergence time. The proposed accelerator deployed on the Zedboard (ZC7020) system-on-chip achieves a low execution time of approximately 4.63μs, 21.3 % speedup, and 3.1 % area reduction compared to the recent particle filter accelerator. The proposed design also demonstrates modularity, achieved through multiple parallel hardware subfilters that provide high throughput for real-time sensor data processing. Furthermore, the proposed accelerator performs a high sampling frequency of 216kHz, making it suitable for high throughput and real-time applications.

INDEX TERMS Pose estimation, particle filter (PF), mobile robotics, localization, pseudorandom number generator (PRNG), cellular automata, field programmable gate arrays (FPGA), very large scale integration (VLSI), Monte Carlo Markov chain (MCMC), sampling importance re-sampling (SIR).

I. INTRODUCTION

The autonomous mobile robot is a system that works in an unexpected and partially unknown environment through a cluster of sensors installed on the platform to find an optimal path, localize itself, and navigate. However, the global localization issue in the mobile robot system is challenging due to the unknown initial pose. One of the primary techniques for sampling from complex probability distributions and guessing a dynamic system's state from sensor readings is the Markov Chain Monte Carlo (MCMC) algorithm. Monte Carlo integration using Markov chains selects samples from a complex distribution and averages them to approach expectations. Particle Filters (PF) are Markov Chain Monte Carlo (MCMC) techniques extensively employed in Bayesian inference to generate samples from complex and high-dimensional posterior distributions. These samples estimate integrals necessary for parameter inference, prediction, and model comparison [1]. The robustness of the MCMC algorithms might lead to slow convergence, as the exploration of the relevant space with significant probability mass can take a long time due to the simulation's tendency to make local jumps near the current position. It is logical to explore methods for expediting the convergence of MCMC algorithms to its stationary distribution, hastening the convergence of MCMC estimates to its expectation, and enhancing the exploration of a given MCMC algorithm within the support of the target distribution.

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria De Munari.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

However, a trade-off exists in developing ultra-performance accelerators versus constrained resources in FPGAs, which means deploying real-time localization tasks on mobile robots is limited despite the added advantages of modularity and accuracy [2]. It is advantageous to optimize performance and hardware utilization to employ resource-constrained devices like field programmable gate arrays (FPGAs) that facilitate parallelism and pipelining in the operations. Using FPGAs makes it feasible to introduce parallelization, enabling real-time hardware implementation for complex Bayesian algorithms while maintaining scalability. State space models in the context of robot localization tasks are the most popular models for modeling the relationships between hidden states and observations defined by unknown parameters, integrating additional complexity into the inference process. A simple example is a robot localization task where the robot has to localize itself and simultaneously map the surroundings in an unknown environment. Since there are numerous domains to which PFs can be deployed, defining and implementing a generic and efficient architecture for all state estimation methods is time-consuming. Conventional resampling approaches are a significant bottleneck for a pipelined system, increasing overall latency [3]. Nevertheless, the Markov Chain Monte Carlo (MCMC) resampling method assists in avoiding the expensive resampling step required by traditional resampling-based solutions.

Our research proposes a scalable and robust source localization model using a mobile robot based on noise-corrupted input sensory measurements. We employed a hardware-efficient Sampling Importance Resampling (SIR) PF technique to elevate the robustness and accuracy of the pose estimation of the robot and its positioning. The issues associated with reducing the computational complexity of PFs are addressed in this research work by adding the memory-efficient cellular automata-based Pseudorandom Number Generator (PRNG) [4] and improved systematic resampler integrated into the SIR particle filter algorithm. The Vose alias-based enhanced systematic resampler is integral to particle filtering by enabling the constant-time generation of sample particles post-alias table initialization, thereby reducing the number of arithmetic operations needed. Furthermore, the inherent parallel architecture improves the performance gains by utilizing multiple processing units simultaneously. It also requires less memory to store precomputed probabilities, making it more efficient for resource-constrained FPGAs. The proposed design achieves higher accuracy while using less execution time and power, making it a potential solution for real-time state estimation robotics applications. The performance of the design is evaluated, compared, and validated using simulations and state-of-the-art implementations. The primary contributions of this paper encompass the following key points:

1) We propose a hardware-based circular buffer mechanism for particle routing between subfilters, a feature that enhances the design's modularity and facilitates seamless data transfers.
2) We implemented a novel area-efficient 32-bit PRNG based on cellular automata, a design that minimizes the utilization of on-chip memory resources.
3) We enhance the systematic resampling process of the particle filter stage by implementing the Vose Alias method to mitigate particle degeneracy issues and exploit the inherently parallel design of Vose Alias on hardware to increase the resampling step's throughput.
4) We implemented the proposed design on a SoC platform, thereby augmenting the system's sampling rate and reducing the overall execution time for processing a substantial number of particles (2k). The proposed architecture is implemented on the low-cost Zynq-7000 SoC. The empirical results of these implementations attest to their efficiency in the context of localization and pose estimation.

The remainder of the research paper is structured as follows: Part II discusses the foundations of Bayesian Inference and Particle Filters, Part III highlights the most recent literature review of particle filter hardware implementation, followed by Section IV, which describes the research methodology. Part V depicts the experiment's outcomes, and Section VI summarizes the paper.

II. BAYESIAN INFERENCE AND PARTICLE FILTERS

Bayesian inference serves as a recursive framework for estimating the probability distribution of unknown variables in a probabilistic model utilizing known data. At its core, Bayesian inference involves calculating the posterior distribution of the unknowns given the data. This process is achieved by multiplying the prior probability of the hypothesis by the likelihood of the data given the hypothesis. The result is then normalized by the marginal probability of the data, yielding the inferred distribution [5]. Generally, the system's state in a discrete-time state-space model is defined as:

$$x_k = f_k(x_{k-1},\ v_{k-1}) \tag{1}$$

where, x_k is system's state at k time step and $v_{k-1}$is noise measurement and $f_k$is some non-linear and time dependent function. The latent variable x_k is assumably hidden and can be calculated by the noisy measurement model defined by equation 2.

$$z_k = h_k(x_k,\ n_k) \tag{2}$$

where, $h_k$is describing the measurement model based on a time-dependent function, and n_k is a noisy vector. The overall aim is to formulate the posterior $p(x_k|z_{1:k})$ of the state space x_k based on the measurement model $z_{1:k}$ till time k, and it is done using prediction and update stages. Using the Chapman-Kolmogorov equation, the prediction probability density function (PDF) of the state at time step k can be

calculated as follows:

$$p(x_k|z_{1:k-1}) = \sum_{x_{k-1}} p(x_k|x_{k-1})p(x_{k-1}|x_{1:k-1}) \quad (3)$$

where $p(x_k|x_{k-1})$ is the motion model defined by the system (1). The PDF acquired from the prediction stage is updated with the range sensor data z_k at time k in the update stage using the Bayesian theorem to develop the posterior distribution:

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{\sum_{x_k} p(z_k|x_k)p(x_k|z_{1:k-1})} \quad (4)$$

where $p(x_k|z_k)$ is described as the likelihood function, and the steps of prediction and update are done recursively while acquiring new measurement data z_k.

Bayesian inference rests on conceptual foundations, including prior probability distribution that represents the degree of belief about the parameters before any observation. The choice of the prior distribution can significantly impact the posterior distribution and the resulting inference. Secondly, the likelihood function describes the probability of observing the data given the model parameters. The likelihood function reflects the data generation method and updates the prior distribution to produce the posterior distribution. Lastly, the posterior probability distribution reflects the updated degree of belief or uncertainty about the parameters [6], [7], [8].

Determining the posterior using the Bayes rule is a methodological perspective analytically determined using standard Kalman filters. Nevertheless, in the case of non-Gaussian settings, obtaining a precise analytical solution becomes intricate. As a result, approximation-based algorithms like Particle Filters (PFs) are utilized to generate an estimated Bayesian outcome. Particle filtering is a generic Markov Chain Monte Carlo sampling technique for inference in state-space models. The system's state updates over time, and knowledge about the state is gained through noisy measurements taken at iterative time steps.

A. SAMPLING IMPORTANCE RESAMPLING PF

One of the most optimal variant of PF is SIR algorithm [1] where, the proposal distribution $p(x_k|x_{k-1}, z_k)$ is assumed to be equivalent to the state transition distribution $p(x_k|x_{k-1})$ and then the re-sampling step is done recursively. Therefore, the update equations in the SIR particle filter are reduced to:

$$x_k^i \sim p(x_k|x_{k-1}^i), \quad (5)$$

$$w_k^i \sim p(z_k|x_k^i). \quad (6)$$

The issue of weight degeneracy can be addressed by iteratively duplicating particles with higher weights and discarding particles with lower weights. Following the resampling step, the particles exhibit a uniform weight distribution approximately that aligns with the desired target distribution $p(x_{0:k}|y_{1:k})$. In the subsequent time step $k+1$, an additional state variable, denoted as x_{k+1}, is introduced

Algorithm 1 SIR Particle Filter Algorithm

Data: x_{k-1}, z_k
Result: x_k
Initialize: For $i = 1, \ldots, N$, set normalized importance weights, $\tilde{w}_0^{(i)} \propto \frac{p(x_0^{(i)})}{q(x_0^{(i)})}$ with $\sum_{i=1}^N \tilde{w}_k^{(i)} = 1$

for $i = 1$ *to* K **: do**
  Propagation: **for** $i = 1$ *to* N **: do**
    particles[i] = sample from $p(x_k|x_{k-1}^{(i)})$ setting new dimension to particle $x_{0:k}^{(i)} = \tilde{x}_{0:k-1}^{(i)}, x_k^{(i)}$
  end for
  Weight Normalization: **for** $i = 1$ *to* N **: do**
    $\tilde{w}_k^{(i)} \propto \frac{p(y_k|x_k^{(i)})(p(x_k^{(i)}|x_{k-1}^{(i)}))}{q(x_0^{(i)})} \times w_{k-1}^{(i)}$
  end for
  Estimation stage: $E_k^{(SIR)} \sum_{i=1}^N f(x_{0:k}^{(i)}) w_k^{(i)}$
  Resampling stage: **if** $N_{eff} \leq cN$ **then**
    resample $\tilde{x}_{0:k}^{(i)}$ with replacement from $x_{0:k}^{(i)}$ set $\tilde{w}_k^{(i)} = \frac{1}{N}$ such that $(\tilde{x}_k^{(i)}, \tilde{w}^{(i)}) = \frac{1}{N}$
  else
    $\tilde{x}_{0:k}^{(i)}, \tilde{w}_k^{(i)} = x_{0:k}^{(i)}, w_k^{(i)}$
  end if
end for

to the particles by sampling from a conditional importance distribution $q_{k+1}(x_{k+1}^{(i)}|x_{0:k}^{(i)})$. Particle filters permit the option of performing resampling optionally. This means that resampling occurs whenever the effective sample size N_{eff} is less than or equal to a specified constant c out of the total number of particles being employed.

Particle filters commonly use $c = 1$, which triggers resampling at each step. However, an alternative value can have advantages, as resampling introduces extra variability into the estimates. When the importance weights exhibit minimal degeneracy, introducing this extra variance through resampling becomes unnecessary. During each iteration, essentially there are two particle approximations: one represented by the set $x_{0:k}^{(i)}, w_k^{(i)}$ before resampling and other by set $\tilde{x}_{0:k}^{(i)}, w_k^{(i)} = 1/N$ after resampling. Although both yield unbiased estimates, the estimator exhibits lower variance before resampling.

The pseudo-code for the SIR particle filter is illustrated in Algorithm 1.

III. RELATED WORK ON PF IMPLEMENTATIONS

Particle Filter-based Monte Carlo Localization (MCL) methods have been extensively employed in robotics to handle the localization problem over the previous decade, as they are regarded as one of the most efficient algorithms to deal with non-linear and asymmetric probability densities [6]. A growing interest in recent years has focused on developing hardware accelerators to speed up particle filter computations

in recent years. One of the earliest works in the domain of hardware prototypes for particle filters was illustrated by Athalye et al. [9]. They developed a generic architecture for sampling and resampling in particle filters on FPGA without exploiting the parallelization capacity of the hardware. With the growing interest in edge computing, it would be beneficial to assess the accelerator's sampling rate and its comparison with other hardware accelerators.

Ye and Zhang [10] proposed an improved FPGA particle filter implementation for radar tracking applications and presented experimental results on a synthetic radar dataset. Although the paper explained the hardware architecture, it did not address comparing the performance of the proposed FPGA architecture with other works and its implementation on actual radar data to evaluate its generality. Mountney et al. [11] presented an FPGA-based implementation of a particle filtering algorithm for Brain Machine Interfaces (BMIs). Their algorithm added strategies to parallelize the state vector and probability estimates, but the evaluation of the proposed design was constrained to a small set of experiments. The paper does not compare the proposed architecture with other existing particle filtering implementations nor present a comprehensive analysis of the proposed approach's performance in various settings.

The paper by Agrawal et al. [12] proposed an FPGA implementation of particle filter-based object tracking in video. While the paper extensively explains the hardware design of the PF-based tracking algorithm, it fails to include a comparison between the suggested FPGA implementation and a software version of the same algorithm. Moreover, evaluating the proposed implementation on different video sequences would be interesting to understand its performance under different conditions and its generality. Miao et al. [13] presented an FPGA-based implementation of Bayesian tracking algorithms for multiple sources of neural activity. The proposed approach is based on a PF algorithm designed to track the locations of multiple neural sources efficiently. They suggested a parallel implementation strategy that uses numerous processing elements (PEs) and a primary central unit (CU). CU handles the resampling process, while PE handles sampling and weight updating. On the other hand, the architecture could be more scalable for large-scale particle processing since the communication overhead between the PE and the CU increases linearly with the number of PEs. Schwiegelshohn et al. [14] introduced the FPGA optimized resampling to parallelize the resampling stage. The architecture design process, including the specific decisions made on the design parameters, algorithms, and implementation details, has to be thoroughly described in the article. Even if the suggested architecture is assessed using a benchmark dataset, it is essential to show its success in a real-world scenario.

Sileshi et al. [15] presented two methods for incorporating PFs on an FPGA: a combined hardware/software approach that utilized a soft-core processor and a dedicated hardware design aimed at reducing execution time. However, the research did not examine how the design variables influenced the performance of the suggested hardware accelerator. Also, the paper mentions that the proposed accelerator uses fixed-point arithmetic but does not investigate the impact of quantization on the accelerator's performance.

Krishna et al. [16] suggested an FPGA implementation of a particle filter that uses parallel processing to process many particle filters concurrently using an additional particle routing step, enabling fast and effective computation. The authors suggest a new resampling method to lessen the computing resources. Although the authors successfully reduced the execution time of the SIR operations by incorporating the multiple sub-filters in their design for 1000 particles, the hardware cost increased significantly. Furthermore, the authors recommended parallel implementation of the systematic re-sampler in the particle filtering step; nonetheless, it remains a black box with an abstract-level explanation only. While recent implementations in the field have been executed on hardware using a variety of frameworks, many of them exhibit notable limitations. Hence, there is a pressing need to develop dedicated hardware solutions equipped to efficiently process a substantial number of particles within predefined time constraints to meet the demanding speed criteria of real-time applications. This paper addresses this critical issue by introducing a high-speed architecture characterized by a high degree of parallelism and easy scalability, enabling it to handle a substantial quantity of particles effectively.

IV. METHODOLOGY

This section summarizes the measurement model regarding the robot localization framework, the architecture and hardware design of the FPGA design, the pseudo-random number generator (PRNG) implementation, the algorithm's improved re-sampling approach, and the integration with the robot localization system.

A. MEASUREMENT MODEL OF PF

The particle filter technique on the mobile robot platform demands dynamic mobility between the onboard sensors and the platform itself [17]. The position of the mobile robot $\hat{X}_k^{Rob}$ is explained by the Cartesian coordinate system in the two-dimensional (2D) plane at time instant k:

$$\hat{X}_k^{Rob} = [X_k^{Rob},\ Y_k^{Rob}]. \tag{7}$$

The pose of the mobile robot along the longitudinal axis is represented by ${\O_k}^{Rob}$ that is the expected bearing of the robot. The range sensor data z_k at k time steps, in binary form, is provided to the measurement model through block random access memories (BRAMs) of the FPGA. Then, given the measurement model (2), it can be integrated with the range sensor data as:

$$z_k = m(x_k) + v_k. \tag{8}$$

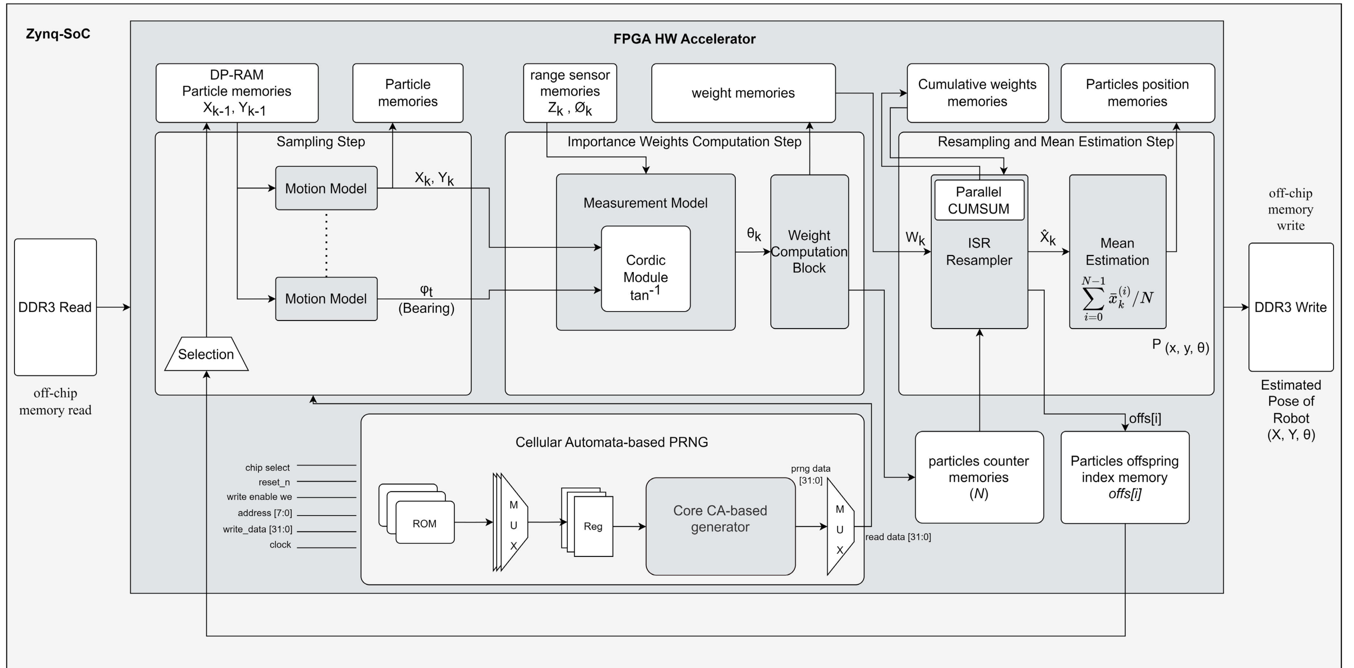


FIGURE 1. FPGA architecture for SIR particle filter.

The expected bearing data of the mobile robot at time step k in the measurement model is given by:

$$m(x_k) = \tan^{-1}\left(\frac{Y_k - Y_k^{Rob}}{X_k - X_k^{Rob}}\right), \tag{9}$$

where, Y_k and X_k are the coordinates of the source sensor in a two-dimensional world. The tangent inverse tan^{-1} function is implemented by hardware efficient CORDIC algorithm for vector rotations, which significantly reduces the number of iterations. To fully harness the FPGA's parallel computing abilities and high-speed architecture, our proposal involves employing four SIR particle filter sub-implementations (*sub-filter F=4*) simultaneously on the FPGA, with each sub-implementation processing 512 particles concurrently. This approach is crucial because the accuracy of the particle filter algorithm relies on the number of particles sampled from the distribution. The initial sampling and importance weights computation stages are essentially parallel since the two stages have no functional dependency. However, the re-sampling stage is sequential as it needs information on all the particles at previous and current time stamps. This sequential nature poses a substantial bottleneck when parallelizing particle filters on FPGA.

To address this challenge, we proposed a particle routing and exchange operation to facilitate the parallelization of the resampling step by efficiently exchanging "s" particles between multiple subfilters. The algorithm initializes a set of subfilters, each containing a collection of particles. It also accepts two parameters: the number of particles to exchange, denoted as s, and the circular buffer size, represented by B. Circular buffers are set up for each subfilter, each having a capacity of B to implement the algorithm. The central part of the algorithm is the particle exchange loop. For every particle marked as "s" for exchange, particles are removed from each subfilter and placed into their corresponding circular buffers. Simultaneously, particles are dequeued from the circular buffer of each subfilter and added to the next subfilter circularly. This process efficiently transfers particles between subfilters while preserving the order. Following the particle exchange loop, each subfilter is updated to contain particles from the circular buffers. The algorithm replaces each subfilter's oldest "s" particles with the particles from their respective circular buffers to ensure that each subfilter now contains particles exchanged between them, as shown in algorithm 2.

B. ARCHITECTURE OVERVIEW OF SIR PARTICLE FILTER

This section presents novel FPGA architecture for the SIR particle filter shown in Figure 1, which exploits the inherent parallelism in each module integrated into the algorithm. The performance of the SIR PF is influenced by different factors, such as the number of particles used, the dimensionality of the state space, and the complexity of the measurement and motion models. Typically, the computational effort of the SIR PF rises linearly with the quantity of particles. If there are N particles, the computation time for each filter iteration will be directly proportional to N. Consequently, the space and time complexity of the SIR particle filter can be expressed as $O(NM)$, where N represents the number of particles and M denotes the complexity of the measurement and motion models. A trade-off between accuracy and computing efficiency will determine the number of particles utilized.

Algorithm 2 Particle Routing Stage

Data: Particle subfilters
[$\text{Subfilter}_1, \text{Subfilter}_2, \ldots, \text{Subfilter}_M$],
Number of Particles to Exchange "s",
Circular Buffer Size B
Result: Updated Particles after Particle Exchange
for $i = 1$ *to* M **do**
  Initialize circular buffer for Subfilter_i with size B.
end for
for $t = 1$ *to* s **do**
  for $i = 1$ *to* M **do**
    Dequeue s particles from Subfilter_i and add them to the circular buffer.
  end for
  for $i = 1$ *to* M **do**
    Dequeue s particles from the circular buffer of Subfilter_i and add them to Subfilter_{i+1}.
  end for
end for
for $i = 1$ *to* M **do**
  Replace the oldest s particles in Subfilter_i with particles from the circular buffer.
end for
Output Updated Particle Subfilters:
[$\text{Subfilter}_1, \text{Subfilter}_2, \ldots, \text{Subfilter}_M$]

Algorithm 3 32-Bit CA-Based Automata

Data: Initialize PRNG with a 32-bit seed
write_data[31 : 0], *address*[7 : 0], 8-bit update rule
Result: *read_data*[31 : 0] PRNG sequence
Function *GeneratePRNG write_data, address, update_rule*:
  Data: Initialize *state*[31 : 0] with *write_data*[31 : 0]
  Data: Initialize *read_data*[31 : 0] with 0
  for *gen* ← 1 **to** 32 **do**
    for *i* ← 1 **to** 31 **do**
      $state[i] \leftarrow UpdateRule30(state[i-1], state[i], state[i+1], update_rule)$
    end for
    $state[0] \leftarrow UpdateRule30(state[31], state[0], state[1], update_rule)$
    $read_data[gen-1] \leftarrow state[31]$
  end for
Function *UpdateRule30*($a, b, c, rule$):
  return $rule[4 \cdot a + 2 \cdot b + c]$

1) CELLULAR AUTOMATA-BASED PRNG

In the domain of VLSI design, a fundamental aspect of cellular automaton (CA) registers involves the utilization of n-bit configurations. Within this context, a critical component is a flip-flop at the index denoted as j. This flip-flop is subject to the influence of a next state (NS) logic, which derives its input from neighboring flip-flops. These neighboring flip-flops are strategically positioned at indices $j2, j1, j+1, j+2$, and the flip-flop is located at index j. The collective behavior of this intricate system is intrinsically governed by a specific CA rule, which prescribes its dynamics. In Field-Programmable Gate Array (FPGA) implementations, a cellular automaton-based PRNG exhibits a noteworthy design, as shown in Algorithm 3. This design incorporates 140 lookup tables (LUTs) and 83 slice registers, significantly reducing the area (LUT + FF) and mitigating reliance on BlockRAMs for storing variables. This particular design strategy in Fig. 2 imparts several advantages, notably low-latency access. Such an attribute enhances critical timing path performance and ensures operations synchronization.

The proposed design of CA operates in two phases: the initialization phase and the update phase. The algorithm begins by initializing the PRNG with the provided seed and the PRNG output sequence as empty. It then proceeds to evolve the CA for 32 generations. In each generation, the state of the CA is updated based on the 8-bit address array for the Rule 30 cellular automaton rule [18], which determines the state transition of each cell based on its neighbors. The rule evaluates the three neighboring cells to compute the next state for each cell. The CA evolution process continues for 32 generations, and in each generation, the leftmost cell of the current state is added to the PRNG output sequence. This process is repeated until all 32 generations are completed, resulting in a 32-bit PRNG sequence. The central element of this algorithm is the UpdateRule30 function, which is responsible for applying Rule 30, a widely recognized cellular automaton rule. Within the UpdateRule30 function, the computation of a cell's new state is determined by considering the values of its three neighboring cells and adhering to a deterministic pattern. The resulting PRNG sequence, which is stored in *read_data*[31 : 0], consists of 32-bit pseudo-random numbers.

2) SAMPLING MODULE

The sampling module creates new particles$\{x_k\}_{i=1}^M$ from a proposal distribution $\{x_{k-1}\}_{i=1}^M$that approximate the true posterior distribution of the state variable given the range sensor measurement data. Moreover, the particle samples $\{x_{k-1}\}_{i=1}^M$are further employed to generate another set of particles $\{x_{k+1}\}_{i=1}^M$of the next time step $k+1$. Therefore, accessing and storing two set of particles $\{x_k\}_{i=1}^M$ and $\{x_{k-1}\}_{i=1}^M$ simultaneously requires efficient memory storage element with depth M. In this research, we propose implementing a single dual-port random access memory (DPRAM) for particle storage with depth M, lowering the total memory needed for particle storage. The particles that are updated throughout the re-sampling procedure are the subsets of the sampled particles. A single DPRAM with pointer variables would be ideal for accessing the particles and reading and writing their

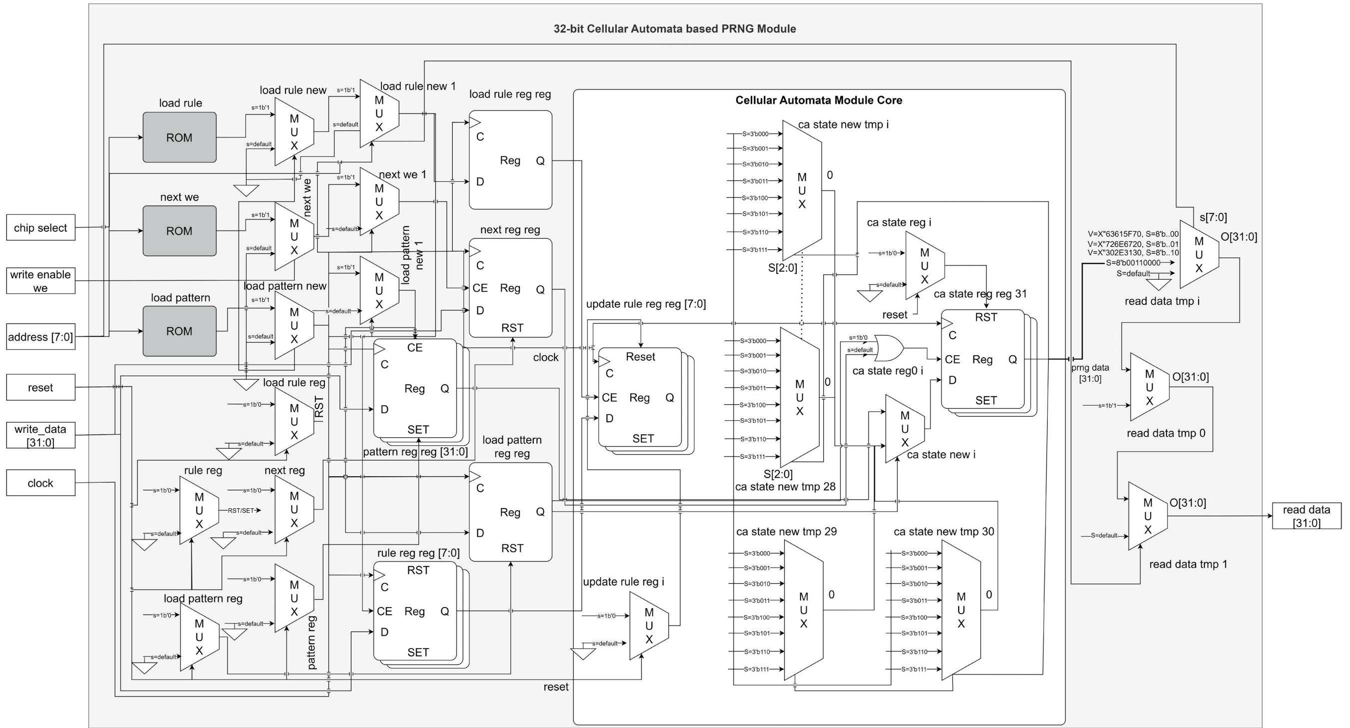


FIGURE 2. Top-level block diagram of CA-based PRNG.

Algorithm 4 Sampling Module of PF for FPGA

Data: Particle from $k-1$, Previous Position $[x^i_{k-1}, y^i_{k-1}]$, and PRNG Seed
Result: Particle at k time x_k (as 32-bit array indices)
Initialize grid parameters: N_{grid} (Number of grid cells in the x and y directions) σ (Standard deviation for random increments) $x_{\max}$ and $y_{\max}$ (Maximum values for x and y)
for $i = 1$ *to* N **do**
  Generate random integer values r_X and r_Y from PRNG Convert r_X and r_Y to float values in the range [0, 1]
  Calculate the increments: $dX = \sigma \times (2 \times r_X - 1)$ $dY = \sigma \times (2 \times r_Y - 1)$
  Update the particle position: $x_k = x_{k-1} + dX$ $y_k = y_{k-1} + dY$
  Map x_k and y_k to grid indices: $x_{\text{index}} = \text{round}(x_k \times \frac{N_{\text{grid}}}{x_{\max}})$ $y_{\text{index}} = \text{round}(y_k \times \frac{N_{\text{grid}}}{y_{\max}})$
  Store x_{index} and y_{index} as 32-bit array indices
end for

indices without repeatedly accessing the replicated particles' memory addresses. An interleaved addressing scheme is employed to access individual particles or state variables within particles. Interleaved addressing distributes particles across memory locations, allowing multiple processing units to access particles concurrently, thereby increasing parallelism and reducing the likelihood of memory contention, potentially improving overall system performance. However, the interleaved addressing scheme requires additional logic to map particles to memory locations, adding complexity to the implementation. The pseudo-code for the sampling module is explained in Algorithm 4.

This module operates by taking into account essential parameters, including N_{grid} (the number of grid cells in the x and y directions), σ (the standard deviation for random increments), and $x_{\max}$ and $y_{\max}$ (the maximum values for x and y). At the algorithm's core is a loop iterating through each particle i from 1 to N, representing the total number of particles in the filter. For each particle, the algorithm generates random integer values r_X and r_Y from a pseudo-random number generator (PRNG), which is fundamental for ensuring stochasticity in the process. The random integer values r_X and r_Y are subsequently converted into float values within the range [0, 1]. This conversion is critical for constraining the increments to the desired range, ensuring a consistent transition from discrete to continuous space. The increments dX and dY are computed based on the generated random values. These increments are drawn from a normal distribution with a mean of 0 and a standard deviation of σ. The random values r_X and r_Y are employed to determine the direction of the increments, effectively mapping them to values between -1 and 1. Particle positions at time k are updated based on the calculated increments as x_k is updated as $(x_{k-1} + dX)$ and y_k is updated as $(y_{k-1} + dY)$. To integrate these updated positions into the grid-based framework, the algorithm maps the continuous positions x_k and y_k to discrete grid indices.

Algorithm 5 Importance Weights Module

Input: Sensor measurements $[z_k^1, z_k^2, \ldots, z_k^N]$, Particle set $S_k = \{(x_k^i, y_k^i, \theta_k^i, x_k^{Rob}, y_k^{Rob}, w_k^i)\}$

for *each particle* $(x_k^i, y_k^i, \theta_k^i, x_k^{Rob}, y_k^{Rob}, w_k^i)$ *in* S_k **do**

  Normalize the particle's weight:

  $w_k^i = \frac{w_k^i}{\sum_i w_k^i}$

  Calculate the angle θ_k between (x_k^i, y_k^i) and (x_k^{Rob}, y_k^{Rob}): $\theta_k^i = \arctan 2(y_k^{Rob} - y_k^i, x_k^{Rob} - x_k^i)$

end for

for *each particle* $(x_k^i, y_k^i, \theta_k^i, w_k^i)$ *in* S_k **do**

  Compute the sector index: $idx(\theta_k^i) = \left\lfloor \frac{4\cdot(\theta_k^i - \phi_k^{Rob})}{\pi} \right\rfloor$

  Compute the weight of the particle based on the measurement and sector index:

  $w_k^i = \text{WeightComputation}(z_k, idx(\theta_k^i))$

end for

Output: Weighted particle set $\hat{S}_k$(updated weights)

for eachparticle $(x_k^i, y_k^i, \theta_k^i, w_k^i)$ *in* S_k **do**

  Store the generated weights w_k^i in the weight memories of the FPGA:

  $\text{WriteToBRAM}(w_k^i, \text{Address}^{(i)})$

end for

This mapping ensures that the particles are aligned with the grid cells. Moreover in mapping, the x_{index} is computed by rounding x_k and scaling it to the grid size of round$(x_k \times \frac{N_{\text{grid}}}{x_{\max}})$. A similar calculation is performed for the y-coordinate to obtain y_{index}. Finally, the calculated x_{index} and y_{index} are stored as 32-bit array indices, effectively incorporating the particles into the grid-based representation.

3) IMPORTANCE WEIGHTS COMPUTATION MODULE

The importance weights computation module's role is to allot weights to particle samples depending on the probability of that particle representing the real state of the system based on sensor readings z_k.

$$w_k = w_{k-1}\, p(z_k|x_k). \tag{10}$$

This is accomplished by comparing the sensor's range data to the projected range values for each particle. The more the weight attributed to a particle, the closer the projected range measurements are to the actual range measurements. In equation (10), the term $p(z_k|x_k)$ is calculated using a custom CORDIC module that estimates the rotation angle (θ_k) of all the particles by calculating an arctan function $tan^{-1}$of the particles' position and robot position in a two-dimensional Cartesian coordinate plane.

$$\theta_k = \tan^{-1}\left(\frac{Y_k - Y_k^{Rob}}{X_k - X_k^{Rob}}\right). \tag{11}$$

In equation (11), the CORDIC implements the arctan function [19]. Tuning the configuration of coarse rotation module in IP settings, the input vectors of particles coordinates (X_k, Y_k) and robot position(X_k^{Rob}, Y_k^{Rob}) are rotated in the range of full circle until the Y component converges to 0. The importance weights computation module is shown in Figure 3. The range sensor vectors are provided to the Importance weights computation module through an AXI-stream interface with a data width of 32 bits. The range measurement data is fed to the sequence generator block to calculate the angle between the particles relative to the robot's pose along the longitudinal axis in radians. Mathematically, it can be calculated as follows:

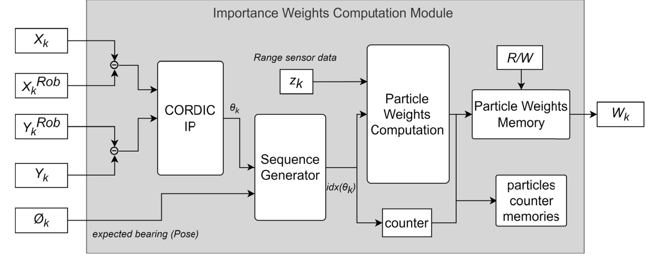


FIGURE 3. Importance weights computation module.

$$idx(\theta_k) = \left\lfloor \frac{4 \times \left(\theta_k - \phi_k^{Rob}\right)}{\pi} \right\rfloor rad. \tag{12}$$

All the particles' relative weights (w_k) are generated by combining the $idx(\theta_k)$ values and range sensor measurements z_k.

4) IMPROVED SYSTEMATIC RE-SAMPLER

Re-sampling is a statistical approach that may be used to address the degeneracy problem. It plays a vital role in particle filtering as it enables the particle set to be refreshed for a more accurate representation of the posterior distribution of the state, considering the available measurement data. Murray et al. [20] introduced systematic re-sampling approaches that computed the cumulative total of particle weights and subsequently reduced the multiplication of the particle count and their cumulative weight. This approach eliminates the need for data dependency within the loop. In this study, we employed the improved version of systematic re-sampling (*ISR*) based on the evidence [21] that it is most optimal for FPGA-based implementations of MCMC applications requiring a large number of particles. It is also preferred due to the ability to minimize Monte Carlo variations while effectively monitoring the effective sample size (ESS), which measures the diversity and quality of the particle set. The weights and total of all particle weights generated by the particle weights computation module remain un-normalized for ISR to process. The output of the re-sampler is the total number of offspring particles of the parent particle $(offs_i,\ i = 1, \ldots, N)$, i.e. the ratio of replication of particle i. The re-sampling goal is to guarantee that the offspring vector $offs_i$ meets the two requirements (13)

and (14).

$$\sum_{i=1}^{N} offs_i = N. \tag{13}$$

$$Exp(offs_i) = \frac{N(w_i)}{Sum\ (w)}. \tag{14}$$

These equations assure that the total quantity of particles (N) remains constant and that the expected value $Exp(offs_i)$ of the replicated particles is proportional to the weights. The pseudo-code for the improved systematic re-sampler is shown in Algorithm 6. The ISR approach requires an initial step that computes the weights' cumulative sum using a parallel cumsum block, *pcumsum*. Although sequential processing is simple, parallelizing it in hardware is challenging due to output data dependence. The three-step recursive doubling approach is more hardware efficient and requires fewer resources for processing large quantities of particles. The algorithm's core principle is to divide the computation into smaller sub-problems that may be solved separately on different processing elements. The architecture relies on several parallel pipelined adders that continuously input new weights in each cycle. An accumulator is positioned after the main data processing path to consolidate the values of each set of weights. P_1-1 uniform adders are also required for CUMSUM to create P_1 outcomes for each weight set, with P_1 denoting the parallel degree for cumulative sums. Meanwhile, the CUMSUM module's overall execution time may be improved from $N + lat_{cumsum}$ in sequential flow to $N/P_1 + lat_{cumsum}$ cycles in parallel flow (i.e., lat_{cumsum} is the latency of CUMSUM datapath).

This approach is entirely parallelizable since there is no requirement for random number generation (PRNG) during this stage, and data dependencies are eliminated throughout loop iterations. The only drawback is that a small amount of extra memory is necessary to store the combined sum of particle weights. The parallel *cumsum's* output is the number of offspring of each particle at prior time steps k ($offs_i$, $i = 1, \ldots, N$), i.e., the particles' replication count, which eventually generates the indices of the re-sampled particles after going through a re-sampler to store in DPRAM. DPRAM provides the advantage of simultaneous read and write operations, which can facilitate parallel processing and efficient access to the particle data as state vector representations.

The algorithm first uses the parallel *cumsum* technique to compute the normalized cumulative sum of the weights. Then, it initializes an empty array called offspring indices of length N (line 2). It sums up and divides each element by the weights to acquire normalized weights. The array *partial_sums* of length *N/2* is initialized with the partial sums of normalized weights in successive pairs. It uses the parallel *cumsum* technique to iteratively compute the cumulative sum of the partial sums until it produces a single value, the total of all the normalized weights. It stores the intermediate cumulative sums in a temporary array called

Algorithm 6 Improved Systematic Resampler

```
Data: Particle weights at k[w_k], N, PRN
Result: offs[i] offspring indices vectors of length N
⇒Normalized cumsum of weights
cumsum_w = p_cumsum(w_k)
initialize offs[i] = 0
A_w = sum[w_k] /w_k[N]
partial_sums[] = ∑A_w
temp_cumsum = sum(A_w)
Create ind[i] ∍ i[1...N/2] → ind[i] = i2
prefix_sum[i] = temp_cumsum[2i]
cumsum_alias=cumsum_w
Normalize cumsum_alias
Create prob[i], alias[i] arrays
for i = 1 to N do
    PRN_fp=PRN[i]×N ⇒fp =
      floating point
    prob[i]=floor(PRN_fp)
    if cumsum_alias[offs[i]]≥prob[i] then
        alias[i]=offs[i]
    else
        alias[i]=j ∀ cumsum_alias[j]≥prob[i],
          ⇒j is smallest index
    end if
end for
if offs[i]==alias[i] then
    keep[i]=1
else
    keep[i]=0
end if
flip[i] = keep[i]
for i = 1 to N do
    if keep[i] == '0' and keep[alias[i]] == '1' then
        set flip[i] = '1'
    end if
end for
Generate binary array flip[2i]
 for i = 1 to N do
    if flip[i] == '1' and flip[2i] == '1' then
        offs[i] = alias[i]
    else
        if flip[i] == '1' and '0' then
            offs[i] = i
        end if
    end if
    if flip[i] == '0' then
        offs[i] = i
    end if
end for
```

temp_cumsum. It creates an array called indices of length *N/2*, where *indices[i] = i2*. It creates an array called *prefix_sums* of length *N/2*, where array *prefix_sums[i]* is equal to *temp_cumsum[2i]*. It recursively computes the cumulative sum of *prefix_sums* using the parallel cumsum method until

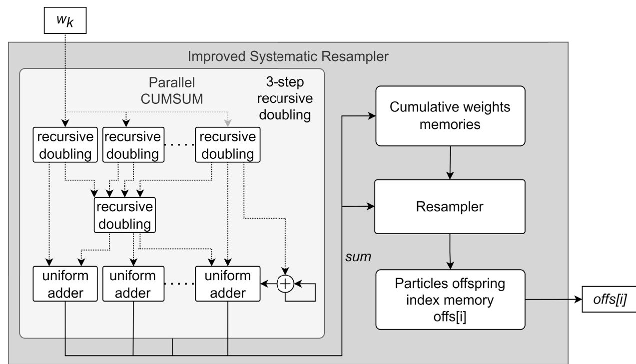


FIGURE 4. Improved systematic resampler.

it gets the array *cumsum*. It then duplicates $cumsum_w$ as $cumsum_{alias}$ and normalizes $cumsum_{alias}$ by dividing each element by the sum of all elements.

It leverages a cellular automata PRNG with the random seed to create N random integers between 0 and 1. It generates two N-dimensional arrays called prob and alias. It multiplies the ith random number by N for each i from 1 to N to get the floating-point value PRN_{fp}. It obtains the integer floor of PRN_{fp} using floor(PRN_{fp}) and puts it in the ith member of offspring indices. The residual fraction of PRN_{fp} is then computed by subtracting floor(PRN_{fp}) from PRN_{fp} and stored in the ith element of prob. If the value of the array $cumsum_{alias}[offs[i]]$ exceeds *prob[i]*, the variable *alias[i]* is assigned the value *offs[i]*. Conversely, if $cumsum_{alias}[j]$ is greater or equal to *prob[i]* for the lowest index *j*, *alias[i]* is set to *j*, and this value is stored in the ith element of the array *alias* (line 17). Subsequently, an array *keep* of length N is generated, where *keep[i]* is 1 if *offs[i]* equals *alias[i]*, and 0 otherwise, designating duplicates as flips. For each *i* from 1 to *N*, if *keep[i]* is 0 and *keep[alias[i]]* is 1, *flip[i]* is set to 1. Employing the pseudorandom number (PRN) with a specified random seed, a binary array named *flip2* is constructed with a 0.5 probability for each member being 0 or 1. If *flip[i]* equals 1 and *flip2[i]* equals 1, *offs[i]* is set to *alias[i]* for each *i* from 1 to *N*. Conversely, if *flip[i]* equals 1 and *flip2[i]* equals 0, *offs[i]* is set to *i*. If *flip[i]* equals 0, *offs[i]* is set to *i*. Lastly, the method generates the offspring indices containing the particle indices chosen as offspring. The initialization time complexity of the Vose Alias technique is $O(n)$, while the cost of sampling once is constant $O(1)$. The block diagram of ISR module is shown in Figure 4. In contrast to other variants of resampling methods, such as roulette wheel selection, stratified and rejected, offspring evaluation can be readily done in parallel with the proposed method.

5) MEAN ESTIMATION MODULE

The SIR particle filter's mean estimation module is used to estimate the mean of a target distribution using a set of particles. To generate the summation, particle locations from the resampler module are input in parallel and aggregated over F cycles (F=4, sub-filters in parallel for implementation). This sum is divided by the entire quantity N of particles by right shifting $log_2(N)$ times to obtain the average [22]. The particle filter iterates to update the estimate as newer data become available. The mean estimates the location of the robot source $P(x, y, \theta)$. Overall, the SIR particle filter's mean estimate module is a practical algorithm for calculating the mean of a target distribution using a set of particles, allowing for accurate and robust estimation of the posterior distribution of a system state in Bayesian filtering. For mobile robot pose estimation, using quaternions to represent particle orientation allows a more accurate and efficient estimation of the robot's pose. The approach can better capture the uncertainty in the robot's position and update the belief distribution over time by employing a population of particles with associated quaternions. This pipeline leads to more robust and accurate localization and posture estimation, especially in noisy and unpredictable situations. The particle's weights with their computed quaternions from the CORDIC IP are summed and normalized to estimate the robot's orientation. This estimate represents the weighted average of the orientations of all the particles, with the weights given by their importance weights.

V. RESULTS AND DISCUSSION

This section will address the utilization of hardware resources of the proposed SIR particle filter on ZC7020, considering the power constraints of the mobile robot system and real-time localization application. We will also compare our findings to those of cutting-edge research works and implementations.

A. TIME EXECUTION ANALYSIS

We utilized four simultaneous SIR filters ($F = 4$) to process 512 particles ($N = 512$) each for hardware implementation on ZC7020. Because the sampling and important weights computation blocks have mutually dependent functions, these modules were pipelined at the architectural level to reduce time execution and latency while increasing design throughput. Although employing pipeline structures requires more hardware resources and increases the design's complexity, it considerably improves efficiency by dividing data processing into stages; each stage may operate independently on a subset of the data. Since the sampling and importance weights computation stage are pipelined, their combined execution time is $N/(F + lat_s + lat_{imp})$, and the vose alias-based ISR resampler takes two times more clock cycles to execute $2N/(F + lat_s + lat_{imp})$. The total clock cycles for ISR resampler with parallel degree P_1 is $N/P_1 + lat_{add} \log_2 P_1 + lat_{adder} + lat_{accum}$ where, lat_{add}, lat_{accum} are the latency of adder and accumulator in the ISR resampler and lat_s, lat_{imp}, and lat_r represents the startup latency of sampling, importance weights computation and improved systematic resampling modules. The total time execution of SIR particle filter to process N particles is estimated to be:

$$T_{simpr} = (\frac{3N}{F} + lat_s + lat_{imp} + lat_r)T_c \quad (15)$$

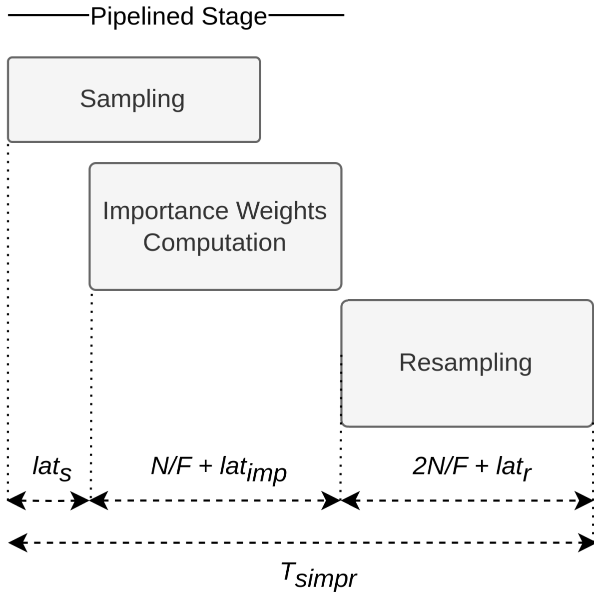


FIGURE 5. Proposed design time execution.

where, T_c is the clock period of the proposed design of the SIR filter. To perform a single iteration of the proposed SIR particle filter, the operation takes T_{simpr} time to complete, as shown in Figure 5.

Figure 6 illustrates the time execution of the proposed SIR particle filter architecture concerning the number of sub-filters (F=4) processing varying quantities of N particles concurrently (N=128/256/1024/2048). If the particle count is low, the particle filter may be unable to adequately reflect the system's potential states, resulting in poor location estimation accuracy. Conversely, if particles are too large, the particle filter's computational complexity may become too high, which is not an option for a low-constrained mobile robot system [23]. Sampling larger particles represents a broader range of possible states, which can help the particle filter explore the state space better, reduce the variance in the importance weights, and find the actual state of the system. Therefore, considering the resource constraints of the hardware, we employed the optimal quantity of particles in our experimental study to keep the system stable.

Adding parallelization to the proposed design by employing additional sub-filters (log_2F) reduces execution time remarkably, but at the cost of increased hardware resources of the FPGA, as depicted in Figure 7. Each sub-filter can independently process *N/F* particles, which allows for parallelization and reduces the execution time notably but comes at the cost of increasing hardware resources. Nevertheless, there is a performance ttrade-offbetween the execution time and resources utilized during architectural synthesis.

The available onboard hardware resources restrict the maximum number of sub-filters mapped on an FPGA, limiting the maximum attainable speed.

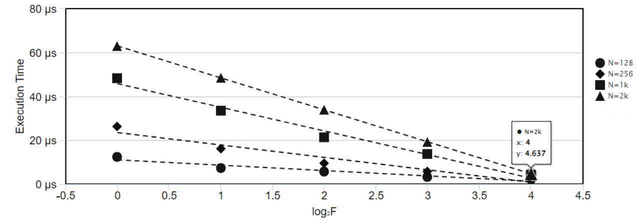


FIGURE 6. Execution time of particle filter vs No.of sub-filters.

B. HARDWARE DESIGN AND RESOURCE UTILIZATION

The proposed architecture of the particle filter accelerator is implemented on Zedboard embedded with a low-cost SoC XC7Z020-CLG484-1, combining two A9 ARM Cortex cores and Artix-7 programmable logic. Regarding computing capability, the board has 85K Logic Cells for implementing various logic functions and sequential logic functions like state machines and counters, 53,200 look-up tables (LUT), 106,400 flip-flops, and 560kb of Block RAM, providing ample onboard memory resources. It has 220 DSP slices for signal processing, which is essential for speeding up complex calculations. The FPGA design of the SIR particle filter was implemented using a single precision floating point arithmetic format in the high-level synthesis tool Vitis HLS 2022.1. Vitis HLS provides a higher-level design process that can increase productivity, reduce development time, and provide performance-optimized hardware designs. It is essential to acknowledge that if the memory bandwidth is not adequate to sustain a continuous flow of data to the processing elements, the performance of the FPGA may be limited when utilizing on-chip FPGA memory to handle a maximum of 2^{12} particles, with the particle weights stored in the respective memories and the need to transfer these weights to or from the memory. The DDR3 memory interface has a maximum operating frequency of 533 MHz with a configured 32-bit data width. The particle transfer between the particle filter accelerator module and DDR memory is enabled by creating a block design. In order to create the interface between the accelerator module and the CPU through the DMA controller, an AXI DMA (direct memory access) IP was used. The AXI DMA was customized, with the stream data width set to 32, aligning with the 32-bit interface of the particle filter accelerator IP in this architecture. The IP is linked to the M_AXIS_MM2S interface through its s_axis. The Zynq PS (processing system) comprises the CPU and DDR memory controller. For our proposed application using a single precision floating point arithmetic datatype, the amount of data that needs to be processed is:

$$12\,B/particle \times 2048 \times 216\,kHz = 5.3\,MB/s. \quad (16)$$

Figure 7 illustrates that with increasing the number of sub-filters (F>1), the hardware resources, i.e., LUTs, slice registers, and BRAM increased proportionally. The graph in Figure 7 shows increased growth in the FPGA LUT resources with increased sub-filter usage. After implementing the proposed design on Zedboard, incorporating

four sub-filters responsible for processing 512 particles each, the utilization includes 10.634 thousand slice lookup tables (LUTs), 8.75 thousand flip-flops, 78 digital signal processors (DSP) employed for multiplication operations during the sampling and weights computation stage, and 66 block random access memory (BRAM) units designated for on-chip storage and expedited particle update, ensuring swift access. Furthermore, our suggested design's resource consumption can be reduced to 1.598k registers and 1.228k LUT with a single sub-filter; however, this would result in a longer execution time. While maintaining four sub-filters, the proposed design exhibits area efficiency by utilizing only 20% of the accessible slice LUTs, 35% of the DSP blocks, and 47% of the BRAM from the available resources on a constrained device. A 16-bit fixed-point representation has been employed for both particles and their corresponding weights in our study. Bearing-related data, encompassing the mobile robot angle and the angles of particles utilized within the importance weights computation block, are expressed using a 12-bit fixed-point representation. A 1.228k Look-Up Table (LUT) suffices for processing 512 particles in a singular sub-filter configuration devoid of parallelization. The temporal expenditure for SIR operations approximates 463 clock cycles. Conversely, a more efficient design employing four sub-filters demonstrates superior efficiency, accomplishing SIR operations in 184 clock cycles at a clock frequency of 100 MHz, yielding a sampling frequency of approximately 543.48 kHz. However, this enhancement comes at the expense of heightened resource utilization, mandating the employment of 10.634k LUTs. The execution time for one set of SIR operations was calculated to be approximately 1.44 microseconds, showcasing the design's efficiency in handling parallelized particle filter computations with high throughput and sampling rates. In the FPGA implementation, the Cellular Automata-Based Pseudo-Random Number Generator (PRNG) module consumes 140 LUTs and 83 flip-flops (FF), contributing to the overall hardware resource utilization. The Sampling and Importance Weight Computation block, comprising pipelined stages, also utilizes 4.524k LUTs and 3.5k FF. Furthermore, the Vose Alias-Based Resampling block, executing over two times the clock cycles of the aforementioned pipelined stages, requires 5.97k LUTs and 5.167k FF. These allocations collectively result in a total hardware resource utilization of 10.634k LUTs and 8.75k FF for the FPGA design.

C. RESAMPLING QUALITY AND LOCALIZATION ESTIMATION

The root-mean-square error (RMSE) measure is used to evaluate the resampling quality of our proposed design (19). RMSE quantifies the average disparity between the predicted state and the actual state of the system. Moreover, the variance of weights quantifies the variability of the resampled particles' weights. A lower variance indicates that the particle weights are distributed evenly, which is excellent for a realistic resampling step. The simulated weights are created with different particle numbers N for experimentation, and their relative variance y ($y = mean(y_{est} - y_{obs})$) is computed, where y_{obs} is the observation value at time k and y_{est} is the estimated values of particles from the sampling module. The RMSE calculations are done for varying quantities of particle filters, i.e., N=2^4 to 2^{12} with a relative variance of 1 (i.e., y=1). The resampling quality of the improved systematic resampler does not change when the parallel degree P_1 is changed, and the RMSE values decrease as the number of particles increases.

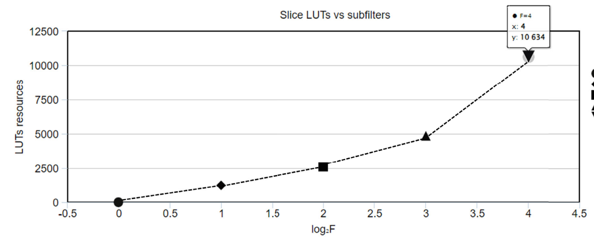


FIGURE 7. Slice LUTs vs No. of sub-filters.

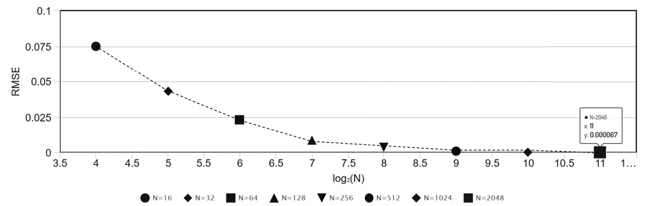


FIGURE 8. RMSE of the ISR resampler for various No. of particles at y=1.

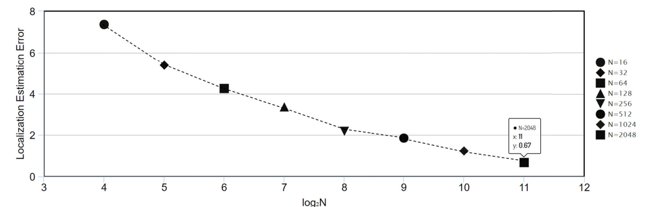


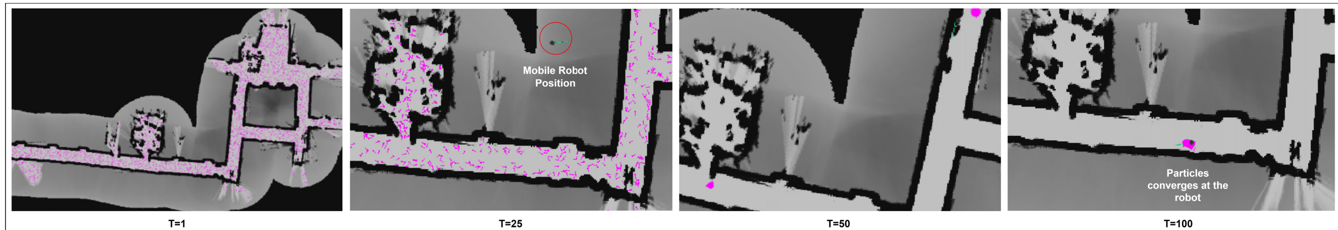
FIGURE 9. Localization estimation error versus the No. of particles.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{offs_i}{N} - \frac{w_i}{sum(w_k)}\right)^2}. \tag{17}$$

The estimation accuracy as a function of the number of particles (N) evaluated using:

$$Estimation\ Error = \sqrt{(P_x - x)^2 + (P_y - y)^2} \tag{18}$$

where, P_x and P_y indicate the estimated location of the mobile robot source obtained by the proposed SIR particle filter, while x and y represent the robot's real position. The localization estimation error obtained by the proposed SIR particle filter is illustrated in Figure 9. Increasing the number of particles significantly reduces the error and converges to an approximate value of 0.67.



2D Localization problem on an occupancy grid map with 2048 particles.

FIGURE 10. Source localization simulation on ROS.

TABLE 1. Comparison with SOTA particle filter implementations on FPGA and performance analysis.

Reference Work	Application	Algorithm	Hardware	Particles (N)	Utilization	Time (μs)	F_s (kHz)
Athalye et al. [9]	Tracking	SIRF	Xilinx Virtex 2 pro	2048	3.85k LUTs, 4.39k FF$^{\psi}$, 13 DSP, 18 BRAM	60.24	16
Ye and Zhang [10]	Radar tracking	SIRF	Xilinx Virtex 5	1024	7.38k LUTs, 13.69k FF, 4 DSP, 13 BRAM	21.74	46
S. Agarwal et al. [12]	Video Object tracking	-	Xilinx Virtex-5	31	59.39k LUTs, 2 DSP, 52 BRAM	23.5	42.55
F. Schwiegelshohn [14]	Position Estimation	Gibbs (Resampler)	Zynq 7020	14	52.41k LUTs, 5.39k FF, 214 DSP	6*	166.67*
B.G.Sileshi et al. [15]	Localization	SIR	Kintex-7	1024	19.12k LUTs, 1.46k FF, 86 DSP, 260 BRAM	55.37	18
A. Krishna et al. [16]	Robot Localization	SIR	Artix-7	1024	10.97k LUTs, 12.35k FF, 16 BRAM	5.62$^{\gamma}$	178$^{\gamma}$
Proposed	Robot Localization	SIR	Zynq 7020	2048	10.634k LUTs, 8.75k FF, 78 DSP, 66 BRAM	4.63$^{\eta}$	216$^{\eta}$

FF^{ψ} = Flip Flops, F_s = sampling frequency, * only resampling stage, $^{\gamma}$ No. of parallel sub-filters (F=8), $^{\eta}$ No. of parallel sub-filters (F=4)

The clock frequency f_{clk} of the proposed design and the data sampling frequency f_{samp} determine the selection of additional sub-filters used for FPGA implementation which is equal to ($f_{samp} = 1/T_{simpr}$). f_{clk} is tuned according to the maximum frequency attained by the design during the synthesis process without producing any negative slack in the design.

D. SIMULATION ON ROBOT OPERATING SYSTEM

We model our design approach in C++ to evaluate our proposed design for the mobile robot localization problem. Furthermore, we use the likelihood fields model to simulate the range sensors since it is more resilient and smooth in cluttered surroundings. This method aims to project the sensor reading endpoints into the coordinate frame of the map. To validate the proposed SIR particle filter algorithm, simulations with a differential drive mobile robot model were run in Gazebo and RVIZ. We used Ubuntu 18 and ROS melodic to simulate the 2D localization environment. A Unified Robot Descriptive Format (URDF) file is created to build a mobile robot. The simulation uses a custom three-wheel mobile robot simulator with onboard laser range sensors and a map. The robot's initial pose is known in the map, and we use the Gaussian distribution to sample 2000 particles. At time step T=1, the particles dispersed over the map, approximating the mobile robot's possible position. The robot's motion model forecasts its location and orientation at each time step as it progresses across the map. The large green arrow represents the heading pose of the mobile robot calculating recursively, while the little pink arrows represent the particle pose with arrows. The estimation of the particle's proximity to the accurate localization of the robot within an occupancy grid map relies on correlating the distances of close objects measured by sensor rays with the distances from the particle to the occupied grid cells, which are determined using geometric processes in the measurement model [24]. The approach involves executing the model for 100 time steps involving 4000 particles. Simultaneously, the mobile robot moves towards the goal specified by the user. By the 100th time step, the particles converge on the robot, exhibiting a minimal estimation error of 0.42, as shown in Figure 10.

E. SOTA COMPARISON

Table 1 compares our proposed architecture to recent state-of-the-art particle filter implementations on FPGA. Most particle filter designs addressed in the literature study need to be more scalable and have limited parallelism, e.g., Athalye et al. [9] architecture is generic. It does not incorporate any parallelization in the design. It contains a low sampling rate of about 16 kHz for 2048 particles, which is approximately 93% slower than the sampling rate of our design. Considering the other methods and applications that are similar to ours, e.g., A. Krishna et al. [16], the state-of-the-art existing methods are 0.8 times lower than our design. Moreover, the resampling stage is the primary bottleneck as it is essentially sequential and requires particle samples

from the previous stages. We proposed a complete parallel resampling stage utilizing parallel cumsum to evaluate the cumulative sum of the particle weights. This operation is extensively parallel and computed efficiently using FPGAs. Our solution is robust, scalable, and has a reasonable level of design complexity. However, most parallel systems encounter issues with scalability because of the substantial communication bottleneck among the concurrent processing units. Our proposed algorithm has comparable hardware resource utilization for additional sub-filters F=4 with a low execution time of 4.63μs and achieves a sampling frequency $f_s = 216$ kHz. The proposed work shows competitive LUT, FF, and BRAM utilization, indicating efficient resource management. The highest sampling frequency among the works indicates more frequent updates and better real-time performance. Overall, the proposed work outperforms most previous implementations regarding resource efficiency, execution time, and sampling frequency.

The fundamental goal of developing a particle filter accelerator is to use the design for real-time applications requiring limited onboard resources and power. Acknowledging the challenges associated with directly comparing the proposed architecture to other recent implementations due to differences in design strategies, application, hardware specifications, particle numbers, and resampling approach, our design overcomes these obstacles by achieving elevated input sampling rates, particularly for an extensive number of particles, through the configurable adjustment of the parallel sub-filters. Nevertheless, it is imperative to emphasize that our proposed design has shown impressive performance in the context of a high sampling rate and low execution time to ensure fair comparisons with the accelerator implementations in recent works [9], [10], [12], [14], [15], [16]. While employing a single particle filter reduces hardware resources, there will be a trade-off between speed and hardware resources used for a given application of a particle filter.

The proposed FPGA design for a particle filter achieves high-performance benchmarks with a 216 kHz sampling frequency, a low 4.63 microsecond execution time, and optimal area utilization of 10.634k slice LUTs. This novel contribution integrates the Vose Alias Resampling method for efficient resampling, a Cellular Automata-based PRNG for deterministic yet pseudo-random number generation, and strategic parallel and pipeline techniques. These techniques ensure parallel processing of particles, reducing execution time and facilitating high-frequency sampling. The design showcases a harmonized approach, leveraging FPGA capabilities for real-time applications with superior throughput and resource efficiency.

VI. CONCLUSION

This paper presents the novel particle filter accelerator, which mainly consists of sampling, importance weight computation, and an improved systematic resampler based on parallel cumsum modules implemented on the ZC7020 SoC with a CA automata-based random number generator, represents a significant achievement in the field of mobile robot localization. This accelerator offers several advantages over traditional software-based implementations, including lower execution time, reduced area consumption, and higher sampling frequency. Particle filters are more robust to modeling errors and can handle time-varying and non-stationary systems more effectively than traditional Bayesian filters, i.e., extended Kalman filters. Nevertheless, when dealing with a large number of particles, the computing cost of the particle filter is high since it needs numerous iterations to update the particle set at each time step. Therefore, the use of particle filters for real-time mobile robot localization is a computational bottleneck. Moreover, even the hardware architecture of the particle filter on FPGA is partially parallel due to the sequential processing of the resampler. To overcome this issue, we proposed the modified version of the standard SIR particle filter with a novel architecture of the resampling module based on a parallel cumulative summation block. The overall design employs additional sub-filters working in parallel fashion to compute multiple particles more efficiently, reducing the overall time execution at the expense of some hardware cost. The low execution time of approximately 4.63μs and low hardware resources of 10.634k LUTs make it a highly efficient solution for real-time applications with strict timing constraints. Additionally, the cellular automata-based random number generator improves the filter's accuracy and stability, enhancing its overall performance. The high sampling frequency of 216kHz enables the particle filter accelerator to process large amounts of data quickly and efficiently, making it ideal for high-speed data processing applications. This research represents a significant step in developing high throughput and efficient hardware accelerators for particle filtering. It can potentially revolutionize the field of accelerating Bayesian filters in the future.

REFERENCES

[1] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[2] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 25, no. 7, pp. 53–82, Jul. 2010.

[3] S.-H. Hong, Z.-G. Shi, J.-M. Chen, and K.-S. Chen, "A low-power memory-efficient resampling architecture for particle filters," *Circuits, Syst. Signal Process.*, vol. 29, no. 1, pp. 155–167, Feb. 2010.

[4] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[5] G. Mingas, L. Bottolo, and C.-S. Bouganis, "Particle MCMC algorithms and architectures for accelerating inference in state-space models," *Int. J. Approx. Reasoning*, vol. 83, pp. 413–433, Apr. 2017.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics* (Intelligent Robotics and Autonomous Agents Series). Cambridge, MA, USA: MIT Press, 2010.

[7] H. Zhou and S. Sakane, "Sensor planning for mobile robot localization—A hierarchical approach using a Bayesian network and a particle filter," *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 481–487, Apr. 2008.

[8] C. Cheng, J.-Y. Tourneret, and X. Lu, "A Rao-Blackwellized particle filter with variational inference for state estimation with measurement model uncertainties," *IEEE Access*, vol. 8, pp. 55665–55675, 2082.

[9] A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP J. Adv. Signal Process.*, vol. 2005, no. 17, Dec. 2005.

[10] B. Ye and Y. Zhang, "Improved FPGA implementation of particle filter for radar tracking applications," in *Proc. 2nd Asian-Pacific Conf. Synth. Aperture Radar*, Oct. 2009, pp. 943–946.

[11] J. Mountney, I. Obeid, and D. Silage, "'Modular particle filtering FPGA hardware architecture for brain machine interfaces," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Aug./Sep. 2082, pp. 4617–4620.

[12] S. Agrawal, P. Engineer, R. Velmurugan, and S. Patkar, "FPGA implementation of particle filter based object tracking in video," in *Proc. Int. Symp. Electron. Syst. Design (ISED)*, Dec. 2012, pp. 82–86.

[13] J. J. Miao, C. Zhang, A. Chakrabarti, and A. Papandreou-Suppappola, "Efficient Bayesian tracking of multiple sources of neural activity: Algorithms and real-time FPGA implementation," *IEEE Trans. Signal Process.*, vol. 61, no. 3, pp. 633–647, Feb. 2013.

[14] F. Schwiegelshohn, E. Ossovski, and M. Hübner, "A fully parallel particle filter architecture for FPGAs," in *Applied Reconfigurable Computing*, A. R. Computing, K. Sano, D. Soudris, M. Hübner, and P. C. Diniz, Eds. Cham, Switzerland: Springer, 2082.

[15] B. G. Sileshi, J. Oliver, and C. Ferrer, "'Accelerating particle filter on FPGA," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 591–594.

[16] A. Krishna, A. van Schaik, and C. S. Thakur, "FPGA implementation of particle filters for robotic source localization," *IEEE Access*, vol. 9, pp. 98185–98203, 2021.

[17] J. L. Palmer, R. Cannizzaro, and B. Ristic, "LTER-based bearings-only tracking algorithm," in *Proc. Australas. Conf. Robot. Autom. (ACRA)*, 2015, pp. 2–3. [Online]. Available: https://www.araa.asn.au/acra/acra2015/papers/pap170.pdf

[18] S. Wolfram. (1983). *Rule 30*. [Online]. Available: https://mathworld.wolfram.com/Rule30.html

[19] Xilinx. (2082). *LogiCORE IP Product Guide*. [Online]. Available: https://www.xilinx.com/products/intellectual-property/cordic.html

[20] L. M. Murray, A. Lee, and P. E. Jacob, "Parallel resampling in the particle filter," *J. Comput. Graph. Statist.*, vol. 25, no. 3, pp. 789–805, Jul. 2016.

[21] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2014, pp. 191–198.

[22] N. T. Briggs, W. H. Slade, E. Boss, and M. J. Perry, "Method for estimating mean particle size from high-frequency fluctuations in beam attenuation or scattering measurements," *Appl. Opt.*, vol. 52, no. 27, pp. 6710–6735, 2082.

[23] M. Ades and P. J. van Leeuwen, "The equivalent-weights particle filter in a high-dimensional system," *Quart. J. Roy. Meteorol. Soc.*, vol. 141, no. 687, pp. 484–503, Jan. 2015.

[24] S. F. A. E. Wijaya, D. S. Purnomo, E. B. Utomo, and M. A. Anandito, "Research study of occupancy grid map mapping method on Hector SLAM technique," in *Proc. Int. Electron. Symp. (IES)*, Sep. 2019, pp. 238–241.

OMER TARIQ received the B.S. degree in electrical engineering from UET, Pakistan, in 2014. He is currently pursuing the Ph.D. degree with the School of Computing, Korea Advanced Institute of Science and Technology. Before this, he was a Senior FPGA Design Engineer with the National Space Agency and the Centre of Excellence in Science and Applied Technologies (CESAT), Pakistan, for seven years. His research interests include parallel computing, deep learning, applied algorithms, autonomous navigation, and FPGA design.

DONGSOO HAN received the B.S. and M.S. degrees in computer science from Seoul National University, Seoul, South Korea, in 1989 and 1991, respectively, and the Ph.D. degree in information science from Kyoto University, Kyoto, Japan, in 1996. He is currently a Professor with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, South Korea. He is leading the Intelligent Service Integration Laboratory. His main research topics are global indoor positioning, pervasive computing, and location-based services (LBS).