This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

FPGA Implementation of Particle Filters for Robotic Source Localization

ADITHYA KRISHNA[1], ANDRÉ VAN SCHAIK[2], (Fellow, IEEE), and CHETAN SINGH THAKUR[1] (Senior Member, IEEE)

[1] Department of Electronic Systems Engineering, Indian Institute of Science, Bangalore, India.
[2] International Centre for Neuromorphic Systems, The MARCS Institute, Western Sydney University, Australia.

Corresponding author: Chetan Singh Thakur (e-mail: csthakur@iisc.ac.in)

ABSTRACT

Particle filtering is very reliable in modelling non-Gaussian and non-linear elements of physical systems, which makes it ideal for tracking and localization applications. However, a major drawback of particle filters is their computational complexity, which inhibits their use in real-time applications with conventional CPU or DSP based implementation schemes. The re-sampling step in the particle filters creates a computational bottleneck since it is inherently sequential and cannot be parallelized. This paper proposes a modification to the existing particle filter algorithm, which enables parallel re-sampling and reduces the effect of the re-sampling bottleneck. We then present a high-speed and dedicated hardware architecture incorporating pipe-lining and parallelization design strategies to supplement the modified algorithm and lower the execution time considerably. From an application standpoint, we propose a novel source localization model to estimate the position of a source in a noisy environment using the particle filter algorithm implemented on hardware. The design has been prototyped using Artix-7 field-programmable gate array (FPGA), and resource utilization for the proposed system is presented. Further, we show the execution time and estimation accuracy of the high-speed architecture and observe a significant reduction in computational time. Our implementation of particle filters on FPGA is scalable and modular, with a low execution time of about 5.62 μs for processing 1024 particles (compared to 64 ms on Intel Core i7-7700 CPU with eight cores clocking at 3.60 GHz) and can be deployed for real-time applications.

INDEX TERMS Particle filters, Field programmable gate array, Bearings-only tracking, Bayesian filtering, Unmanned ground vehicle, Hardware architectures, Real-time processing.

I. INTRODUCTION

Emergency response operations such as disaster relief, and military applications often require localization of a contaminant chemical or biological source in an unknown environment. Unmanned vehicles are gaining popularity in such applications in recent times due to reduced human involvement and the ability to carry out the task remotely. These autonomous systems can eventually supplement human intervention in various safety-critical and hazardous missions. Nevertheless, conditions in which these missions are conducted vary drastically depending on the environmental factors that result in the sensor receiving noise-corrupted measurements. This poses a significant challenge to unmanned vehicles to navigate and locate a target in an unknown environment autonomously.

Our study demonstrates autonomous source localization using an Unmanned ground vehicle (UGV) and proposes a novel source localization model for light source localization with noise-corrupted input measurements as a proof-of-concept. The model presented here uses a particle filter algorithm [1] to increase the robustness to false detections and noise-corrupted measurements. In recent times, there is a growing popularity of particle filters (PFs) in signal processing and communication applications to solve various state estimation problems like tracking [2], localization, navigation [3], and fault diagnosis [4]. PFs have been applied for models described using a dynamic state-space approach comprising a system model representing the state evolution and a measurement model representing the noisy measurements of the state [5]. In most real-time scenarios, these models are non-linear and non-Gaussian. Traditional filters like Kalman filters prove to be less reliable for such applications, and it is proven that PFs outperform conventional filters in such scenarios [6].

PFs are inherently Bayesian in nature, intending to construct a posterior density of the state (e.g., the location of a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

target or source) from observed noisy measurements. In PFs, the posterior of the state is represented by a set of weighted random samples known as particles. A weighted average of the samples gives the state estimate (location of the source). PFs use three major steps: Sampling, Importance, and Re-sampling for state estimation, thus deriving the name SIR filter. In the sampling step, particles from the prior distribution are drawn. The importance step is used to update the weights of particles based on input measurements. The re-sampling step prevents any weight degeneracy by discarding particles with lower weights and replicating particles having higher weights. Since PFs apply a recursive Bayesian calculation, all particles must be processed for sampling, importance, and re-sampling steps. Then, the process is repeated for the next input measurement, resulting in enormous computational complexity. Further, the execution time of PFs is proportional to the number of particles, which inhibits the use of PFs in various real-time applications wherein a large number of particles need to be processed to obtain a good performance. Several implementation strategies have been proposed in the literature to address this issue and make PFs feasible in real-time applications discussed in Section. II.

A. OUR CONTRIBUTIONS

The contributions of this paper are on algorithmic and hardware fronts:

1) Algorithmic Contribution

We propose a novel source localization model employing a light source as the target/source to be localized and an UGV carrying an array of photodiodes to sense and localize the source. Photodiode measurements and the UGV position are processed to estimate the bearing of the light source relative to the UGV. Based on the bearing of the light source, we try to localize the source using the PF algorithm. Reflective objects and other stray light sources are also picked by the sensor (photodiodes), leading to false detections. In this study, we have successfully demonstrated that our PF system is robust to noise and can localize the source even when the environment is noisy. We introduce two parameters α and β to model the sensor imperfections and background noise activity, respectively. However, the PFs are computationally very expensive, and the execution time often becomes unrealistic using a traditional CPU-based platform. The primary issue faced during the design of high-speed PF architecture is the parallelization of the re-sampling step. The re-sampling step is inherently not parallelizable as it needs the information of all particles. We propose a modification to the standard SIR filter (cf. Algorithm 1) to address this problem and make a parallel and high-speed implementation possible. The modified algorithm proposed (cf. Algorithm 2) uses a network of smaller filters termed sub-filters, each processing independently and concurrently. The processing of total N particles is partitioned into K sub-filters so that at most N/K particles are processed within a sub-filter. This method reduces the overall computation time by a factor of K. The modified algorithm also introduces an additional particle routing step (cf. Algorithm 2), which distributes the particles among the sub-filters and makes the parallel implementation of re-sampling possible. The particle routing step is integrated with the sampling step in the architecture proposed and does not require any additional time for computation. We also compare the estimation accuracy of the standard algorithm with the modified algorithm in Section VIII-C and infer that the modified and the standard approaches do not vary significantly in terms of estimation error. Additionally, the modified algorithm achieves a very low execution time of about 5.62 μs when implemented on FPGA, compared to 64 ms on Intel Core i7-7700 CPU with eight cores clocking at 3.60 GHz for processing 1024 particles and outperforms other state-of-the-art FPGA implementation techniques.

2) Hardware Contribution

We implemented the modified SIR algorithm on an FPGA and key features of the proposed architecture are:

- Modularity: We divide the overall computation into multiple sub-filters, which process a fixed number of particles in parallel, and the processing of the particles is local to the sub-filter. This modular approach makes the design adaptable and straightforward as it allows us to customize the number of sub-filters in the design depending on the sampling rate of input measurement and the amount of parallelism needed.
- Scalability: Our architecture can be scaled easily to process a large number of particles without increasing the execution time by using additional sub-filters.
- Design complexity: The proposed architecture relies on the exchange of particles between sub-filters. However, communication and design complexity increase proportionally with the number of sub-filters used in the design. In our architecture, we employ a simple ring topology to exchange particles between sub-filters to reduce complexity and design time.
- Memory utilization: The sampling step uses particles from the previous time instant to estimate the particles of the present time instant. This requires the sampled and re-sampled particles to be cached in two separate memories. The straightforward implementation of the modified SIR algorithm needs $2 \times K$ memory elements each of depth M for storing the sampled and re-sampled particles for K sub-filters. Here, M is the number of particles in a sub-filter ($M = N/K$). However, applications involving non-linear models require a large number of particles [7]. This would make the total memory requirement $2 \times K$ significant for large K or M. The proposed architecture reduces this memory requirement to K memory elements each of depth M using a dual-port ram, as explained in Section VII-A1. Therefore, the proposed architecture lowers memory utilization, and reduced memory access makes it more energy-efficient.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

- Real-time: Since all sub-filters operate in parallel, the execution time is significantly reduced compared to that of other traditional implementation schemes that use just one filter block [8]. Our implementation has a very low execution time of about 5.62 μs (i.e., a sampling rate of 178 kHz) for processing 1024 particles and outperforms most state-of-the-art implementations, allowing real-time deployment.
- Flexibility: The proposed architecture is not limited to a single application, and the design can be easily modified by making slight changes to the architecture for other PF applications.

The architecture was successfully implemented on the Artix-7 FPGA and the experimental results show its efficacy in source localization.

The rest of the paper is organized as follows: We provide the theory behind Bayesian filtering and PFs in Section III and IV, respectively. An experimental setup for the proposed source localization model using a Bearings-only tracking (BOT) framework is presented in section V. In this framework, input to the filter is a time-varying angle (bearing) of the source, and each input is processed by the PF algorithm implemented on hardware to estimate the source location. Further, in section VI, we propose algorithmic modifications to the existing PF algorithm that make the high-speed implementation possible. The architecture for implementing PFs on hardware is provided in Section VII. Evaluation of resource utilization on the Artix-7 FPGA, performance analysis in terms of execution time, estimation accuracy, and the experimental results are provided in Section VIII.

II. STATE-OF-THE-ART

The first hardware prototype for PFs was proposed by Athalye et al. [8] by implementing a standard SIR filter on an FPGA. They provided a generic hardware framework for realizing SIR filters and implemented traditional PFs without parallelization on FPGA. As an extension to [8], Bolic et al. [9] suggested a theoretical framework for parallelizing the re-sampling step by proposing distributed algorithms called Re-sampling with Proportional Allocation (RPA) and Re-sampling with Non-proportional Allocation (RNA) of particles to minimize execution time. The design complexity of RPA is significantly higher than that of RNA due to non-deterministic routing and complex routing protocol. Though the RNA solution is preferred over RPA for high-speed implementations with low design time, the RNA algorithm trades performance for speed improvement. Agrawal et al. [10] proposed an FPGA implementation of a PF algorithm for object tracking in video. Ye and Zhang [11] implemented a SIR filter on the Xilinx Virtex-5 FPGA for bearings-only tracking applications. Sileshi et al. [12]–[14] suggested two methods for implementation of PFs on an FPGA: the first method is a hardware/software co-design approach for implementing PFs using MicroBlaze soft-core processor, and the second approach is a full hardware design to reduce execution time. Velmurugan [15] proposed an FPGA implementation of a PF algorithm for tracking applications without any parallelization using the Xilinx system generator tool. Schwiegelshohn et al. [16] proposed the FPGA optimized re-sampling (FO-resampling) to parallelize the re-sampling step by introducing virtual particles. A fixed number of virtual particles are generated around every real particle, and if the importance factor (weight) of the real particle is less than the virtual particle, then it gets replaced. Otherwise, the same real particles are propagated in the next iteration. However, the resource utilization of their architecture is substantially higher compared to the conventional PF algorithms. Mountney et al. [17] proposed a modular PF architecture for Brain Machine Interfaces (BMI). Their architecture introduces multiple particle processors to parallelize the state vector and likelihood estimations. Although the state vector estimation and likelihood computations are parallelized, the re-sampling step is done sequentially, which is the major drawback of the architecture. Recently, Alam et al. [18] proposed an improved re-sampling architecture by introducing a weight pre-fetch mechanism to reduce the latency of the re-sampling step. In this technique, new particle weights are pre-fetched along with the random values concurrently, which help in reducing the total number of cycles for re-sampling. Pre-fetching parameters, on the other hand, necessitates the use of additional buffers to store the pre-fetched data, resulting in the increased area and power consumption. Miao et al. [19] proposed a parallel implementation scheme for PFs using multiple processing elements (PEs) and a central unit (CU) to reduce the execution time. PE performs sampling and weight update, while CU performs re-sampling. The communication overhead between the PE and the CU, on the other hand, grows linearly with the number of PEs, rendering the design unscalable for large-scale particle processing. In other work, Velmurugan et al. [20] took an analog approach to implement PF with low-power consumption. Their implementation utilizes a minimum number of data converters to reduce both area and power. However, owing to the analog mixed mode implementation, their architecture is not scalable, and verification of the design is difficult compared to the digital counterparts due to lack of standard design and test flows in large analog implementation.

Further, several real-time software-based implementation schemes have been proposed with the intent to reduce computational time. Hendeby et al. [21], [22] proposed the first Graphical Processing Unit (GPU) based PFs, demonstrating that the GPU-based architecture outperforms the CPU-based implementation in terms of processing speed. Murray et al. [27] provided an analysis of two alternative schemes for the re-sampling step based on Metropolis and Rejection samplers to reduce the overall execution time. They compared it with standard Systematic resamplers [28] over GPU and CPU platforms. Chitchian et al. [23] devised an algorithm for implementing a distributed computation PF on GPU for fast real-time control applications. Zhang et al. [29] suggested an architecture for efficiently implementing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

TABLE 1: State-of-the-art PF implementations. [New Table]

Implementation	Related Works	Algorithm	Device	Application	Remarks
FPGA based digital implementation	Athalye et al. [8]	SIR with systematic re-sampling	Virtex II pro	Tracking	Sequential implementation with low design complexity.
	Bolic et al. [9]	SIR with RPA & RNA	Virtex II pro	Tracking	Parallel implementation utilizing multiple concurrent PEs. High design complexity & not easily scalable.
	Agarwal et al. [10]	Color histogram based PF	Virtex-5	Object tracking in video	Sequential implementation of PF for object tracking in video.
	Ye and Zhang [11]	SIR with residual systematic re-sampling	Virtex-5	Radar Tracking	Sequential implementation with low design complexity.
	Sileshi et al. [12]–[14]	SIR with Independent Metropolis Hasting (IMHA) re-sampling	Kintex-7	SLAM	Proposed HW/SW co-design & fully hardware solutions for PF implementation. No parallelism incorporated in the design.
	Velmurugan [15]	SIR with modified residual systematic re-sampling algorithm	Virtex II pro	Tracking	Sequential implementation using a system generator tool with high resource utilization.
	Schwiegelshohn et al. [16]	FPGA optimized re-sampling (FO-resampling)	Zynq-7020	Position estimation	Parallel implementation with limited scalability and high resource utilization.
	Mountney et al. [17]	Bayesian auxiliary particle filter algorithm (BAPF)	Not provided	Brain Machine Interfaces	State estimation and likelihood computation are parallelized, however, re-sampling is done sequentially.
	Alam et al. [18]	Multinomial re-sampling	Virtex-6	Generic	Improved re-sampling of particle filter design that included a weight pre-fetch function to reduce the re-sampling step's latency.
	Miao et al. [19]	Probability hypothesis density filtering	Virtex-5	Tracking of multiple sources of neural activity	Parallel implementation utilizing multiple PEs. Sampling and weight update is done inside a PE and resampling within a CU. High communication overhead between PE and CU. Not easily scalable.
	This work	Modified SIR with systematic re-sampling	Artix-7	Source Localization	Parallel and pipelined architecture which is highly scalable with low design complexity and comparable resource utilization.
Mixed mode Analog Implementation	Velmurugan [20]	SIR	ASIC (0.35 μm CMOS process)	Target tracking	Re-sampling is implemented in the digital domain and the remaining stages in the analog domain. It consumes very low power. However, not easily scalable.
Graphical Processing Unit (GPU)	Hendeby et al.[1] [21], [22], Chitchian et al.[2] [23], Gong et al.[3] [24], Par et al.[4] [25], Kim et al.[5] [26]	Proposed modified algorithms to implement particle filters efficiently on a GPU.	GeFORCE GTX 7900 [1], GTX 580[2,4], Quadro FX 5800[3], Jetson TX1[5]	More generic in nature.	Parallelized the operation by utilizing multiple cores available on GPU.
	Murray et al. [27]	PF with Metropolis and Rejection resamplers	NVIDIA K20 GPU	Tracking	Showed the implementation of Metropolis and Rejection samplers on GPU and compared them with Systematic resamplers.

PFs on a DSP for wireless network tracking applications. Gong et al. [24] present a shared-memory systematic re-sampling (SMSR) algorithm to parallelize the re-sampling step on a GPU. Their algorithm is very challenging to implement on an FPGA due to the use of shared memory architecture and parallel scan step to obtain the prefix sum. Furthermore, they don't present any architecture for implementing the algorithm on hardware. Choppala et al. [30] introduced a random network as a fixed re-sampling unit in PF. This network assigns each particle a predetermined set of other particles with which it will interact, and the re-sampler randomly selects one particle from the set. However, they don't show the hardware feasibility of the proposed network on FPGA. Par et al. [25] present a parallel implementation of PF algorithm based on both multi-core processors and on a GPU using Compute Unified Device Architecture (CUDA). Their performance analysis shows that up to 75x speedup can be achieved on a 512-core GPU over sequential implementation. Kim et al. [26] implemented PF on a GPU for target position estimation and parallelized the calculation process utilizing multiple GPU cores. The proposed algorithm was simulated on a CPU in MATLAB and then verified on GPU, resulting in a 55% reduction in execution time. However, they do not

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

show the hardware feasibility. In addition, these software-based methods have their own drawbacks when it comes to hardware implementation owing to their high computational complexity. Therefore, it is essential to develop a high-speed and dedicated hardware design with the capacity to process a large number of particles in specified time to meet the speed demands of real-time applications. This paper addresses this issue by proposing a high-speed architecture that is massively parallel and easily scalable to handle a large number of particles. The benefits of the proposed architecture are summarized in Section. I-A2.

III. BAYESIAN FRAMEWORK

The evolution of the state sequence x_t in a dynamic state space model is characterised by:

$$x_t = f_t(x_{t-1}, w_t) \tag{1}$$

where, f_t is a nonlinear function of the state x_{t-1}, and w_t represents the process noise. The objective is to recursively estimate the state x_t based on a measurement defined by:

$$z_t = g_t(x_t, v_t) \tag{2}$$

where, g_t is a nonlinear function describing the measurement model, and z_t is the system's observation vector corrupted by measurement noise v_t at time instant t.

From a Bayesian standpoint, the objective is to construct the posterior $p(x_t|z_{1:t})$ of the state x_t from the measurement data $z_{1:t}$ up to time t. By definition, the posterior is constructed in two stages: prediction and update.

The prediction stage uses the system model (cf. Eq. 1) to estimate a prediction probability density function (PDF) of the state at time instant t, using the Chapman-Kolmogorov equation:

$$p(x_t|z_{1:t-1}) = \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1}) \tag{3}$$

where, the transition probability $p(x_t|x_{t-1})$ is defined by the system model (cf. Eq. 1).

In the update stage, the measurement data z_t at time step t is used to update the PDF obtained from the prediction stage using Bayes rule, to construct the posterior:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{\sum_{x_t} p(z_t|x_t)p(x_t|z_{1:t-1})} \tag{4}$$

where, $p(z_t|x_t)$ is a likelihood function defined by the measurement model (cf. Eq. 2).

The process of prediction (cf. Eq. 3) and update (cf. Eq. 4) are done recursively for every new measurement z_t. Constructing the posterior based on Bayes rule is a conceptual solution and is analytically estimated using traditional Kalman filters. However, in a non-Gaussian and non-linear setting, the analytic solution is intractable, and approximation-based methods such as PFs are employed to find an approximate Bayesian solution. A detailed illustration of the Bayesian framework and its implementation for estimating the state of a system is provided by Thakur et al. [31].

IV. PARTICLE FILTERS BACKGROUND

The core principle behind PFs is to represent the required posterior density with a collection of random samples called particles, each with its own weights, and then calculate the state estimate using these particles and weights. The particles and their weights are represented by $\{x_t^i, w_t^i\}_{i=1}^N$, where N is the total number of particles. x_t^i denotes the i^{th} particle at time instant t. w_t^i represents the weight corresponding to the particle x_t^i. The variant of PF called sampling, importance, and re-sampling filter (SIRF) is presented in Algorithm 1.

Algorithm 1 : SIR Algorithm

Initialization: Set the particle weights of the previous time step to 1/N, $\{w_{t-1}^i\}_{i=1}^N = 1/N$.
Input: Particles from previous time step $\{x_{t-1}^i\}_{i=1}^N$ and measurement z_t.
Output: Particles of current time step $\{\widehat{x}_t^i\}_{i=1}^N$.
Method:

1: **Sampling and Importance:**
2: for $i = 1$ to N do
3: Sample $x_t^i \sim p(x_t|x_{t-1}^i)$
4: Calculate $w_t^i = w_{t-1}^i p(z_t|x_t^i)$
5: end for
6: **Re-sampling:** Deduce the re-sampled particles $\{\widehat{x}_t^i\}_{i=1}^N$ from $\{x_t^i, w_t^i\}_{i=1}^N$.

In the sampling step, particles are drawn from the prior density $p(x_t|x_{t-1}^i)$ to generate particles at time instant t. $p(x_t|x_{t-1}^i)$ is deduced from (1). Intuitively, it can be thought as propagating the particles from time step $t-1$ to t. The sampled particles at time instant t is denoted by $\{x_t^i\}_{i=1}^N$. At time instant 0, particles are initialized with prior distribution to start the iteration. These particles are then successively propagated in time.

The importance step assigns weights to every particle x_t^i based on the measurement z_t. By definition, the weights are given by:

$$w_t^i = w_{t-1}^i p(z_t|x_t^i) \tag{5}$$

However, weights of the previous time step are initialized to $1/N$ i.e $w_{t-1}^i = 1/N$. Thus, we have:

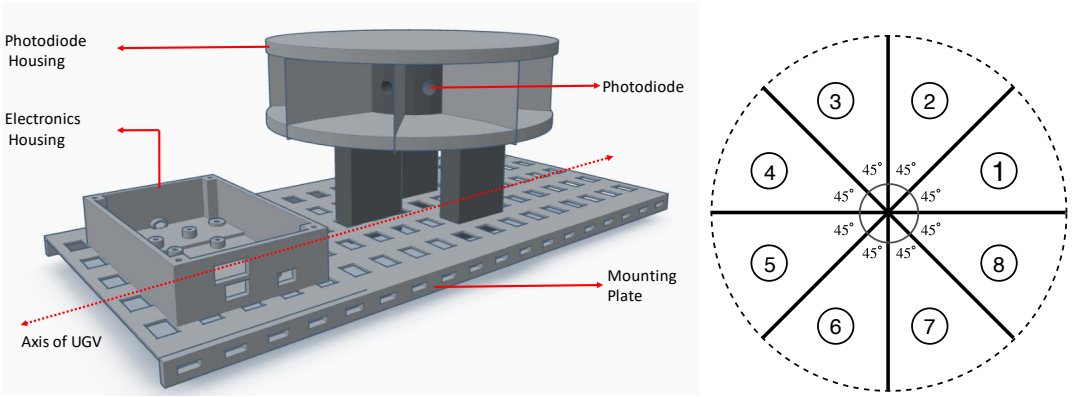
$$w_t^i \propto p(z_t|x_t^i) \tag{6}$$

The re-sampling step is used to deal with the degeneracy problem in PFs. In the re-sampling step, particles with lower weights are eliminated, and particles with higher weights are replicated to compensate for the discarded particles depending on the weight w_t^i associated with the particle x_t^i. The re-sampled set of particles is denoted by $\{\widehat{x}_t^i\}_{i=1}^N$.

V. SOURCE LOCALIZATION MODEL

This section gives an overview of the experimental setup and measurement model relevant to the source localization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access



(a) (b)

FIGURE 1: UGV Design. (a) Schematic of the UGV with a photodiode housing mounted on top. (b) The region around the UGV is divided into 8 sectors with 45° angular separation.

A. OVERVIEW OF THE EXPERIMENTAL SETUP

In our source localization model, an omnidirectional light source serves as a source to be localized. A photodiode housing mounted on top of the UGV (cf. Fig. 1(a)) constitutes a sensor to measure the relative intensity of light in a horizontal plane. The space around the UGV is divided into 8 sectors with 45° angular separation, as shown in Fig. 1(b), and an array of 8 photodiodes are placed inside the circular housing to sense the light source in all directions. The housing confines the angle of exposure of the photodiode to 45°. Depending on the light incident on each photodiode, we consider the output of the photodiode to be either 0 or 1.

The PF algorithm applied to the BOT model requires dynamic motion between the sensor and source [32]. In our experimental configuration, we have a stationary source and a moving sensor mounted on the UGV. The UGV is made to traverse in the direction of the source and eventually converges at the source location. Reflective sources and other stray light sources are potential sources of noise picked up by the sensor, producing false detections. A target-originated measurement, along with noise, is sensed by the photodiodes and processed in addition to the UGV position data to measure the light source's bearing with respect to the UGV. Based on the bearing of the light source, we try to estimate its position using the PF algorithm.

B. MEASUREMENT MODEL

The position of UGV (x_t^{UGV}) at time instant t is defined by the Cartesian co-ordinate system:

$$x_t^{UGV} = [X_t^{UGV}, Y_t^{UGV}]$$

The orientation of the longitudinal axis of UGV is represented by ϕ_t^{UGV}, which gives its true bearing.

The source is considered to be stationary, and its co-ordinates in the 2-dimensional setting is given by:

$$x_t = [X_t, Y_t] \tag{7}$$

At time instant t, a set of 8 photodiode measurements are captured $z_t = \{z_t^1, z_t^2 \cdots z_t^8\}$, which comprise of the target-associated measurement and clutter noise. Then, based on the measurement model (2), the source-associated measurement can be modelled as:

$$z_t = g(x_t) + v_t \tag{8}$$

Since the measurement gives the bearing information of the source, we have:

$$g(x_t) = tan^{-1}\left(\frac{Y_t - Y_t^{UGV}}{X_t - X_t^{UGV}}\right) \tag{9}$$

The four-quadrant inverse tangent function evaluated from $[0, 2\pi)$ gives the true bearing of the source.

The relevant probabilities needed to model the sensor imperfections and clutter noise are as follows:

(i) The probability of clutter noise (n_t) produced by a stray or reflective light source is: $p(n_t) = \beta$.

(ii) The probability of the j^{th} photodiode output being 1 i.e., $(z_t^j = 1)$ either due to the light source or clutter noise is: $p(z_t^j | x_t, n_t) = \alpha$.

(iii) If there is a light source in the sector j, then j^{th} photodiode output will be 1 with a probability of α irrespective of noise. The likelihood of photodiode output being 1 or 0 in the presence of the source is:

$$p(z_t^j | x_t) = \begin{cases} \alpha, & \text{for } z_t^j = 1. \\ 1 - \alpha, & \text{for } z_t^j = 0. \end{cases} \tag{10}$$

(iv) If there is no source in sector j, then there is a noise source with probability β. The likelihood of photodiode output being 1 or 0 in the absence of the source is:

$$p(z_t^j | \widetilde{x_t}) = \begin{cases} \alpha\beta, & \text{for } z_t^j = 1. \\ 1 - \alpha\beta, & \text{for } z_t^j = 0. \end{cases} \tag{11}$$

These two likelihoods are used in our system to model the sensor imperfections and noise, and even with high noise probability β, the PF algorithm is robust enough to localize the source.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

VI. ALGORITHMIC MODIFICATION OF SIRF FOR REALIZING HIGH-SPEED ARCHITECTURE

In this section, we suggest modifications to the standard SIR algorithm to make it parallelizable. The key idea of high-speed architecture is to utilize multiple parallel filters, termed sub-filters, working simultaneously and performing sampling, importance, and re-sampling operations independently on particles. The architecture utilizes K sub-filters in parallel to process a total of N particles. Thus, the number of particles processed within each sub-filter is $M = N/K$. In comparison to traditional filters, the amount of particles processed inside each sub-filter is reduced by a factor of K.

Algorithm 2 : High-level description of each sub-filter k performing SIR and particle routing operations.

Initialization: Set the particle weights of previous time step to 1/M, $\{w_{t-1}^{(k,i)}\}_{i=1}^{M} = 1/M$.
Input: Particles from previous time step $\{x_{t-1}^{(k,i)}\}_{i=1}^{M}$ and measurement z_t
Output: Particles of current time step $\{\widehat{x}_t^{(k,i)}\}_{i=1}^{M}$
Method:

1: **Particle Routing:** Exchange Q particles with neighbouring sub-filters.
2: $\{x_{t-1}^{(k,q)}\}_{q=1}^{Q} \leftarrow \{x_{t-1}^{(k-1,q)}\}_{q=1}^{Q}$ for $k = 2, \cdots K$, and
3: $\{x_{t-1}^{(k,q)}\}_{q=1}^{Q} \leftarrow \{x_{t-1}^{(K,q)}\}_{q=1}^{Q}$ for k=1.
4: **Sampling and Importance:**
5: for $i = 1$ to M do
6: Sample $x_t^{(k,i)} \sim p(x_t|x_{t-1}^{(k,i)})$
7: Calculate $w_t^{(k,i)} = w_{t-1}^{(k,i)} p(z_t|x_t^{(k,i)})$
8: end for
9: **Re-sampling:** Compute the re-sampled particles $\{\widehat{x}_t^{(k,i)}\}_{i=1}^{M}$ from $\{x_t^{(k,i)}, w_t^{(k,i)}\}_{i=1}^{M}$.

The sampling and importance steps are inherently parallelizable since there is no data dependency for the particle generation and weight calculation. However, the re-sampling step cannot be parallelized as it needs to have the information of all particles. This creates a major bottleneck in the parallel implementation scheme. Thus, in addition to the SIR stage, we introduce a particle routing step, as shown in Algorithm 2, to route particles between sub-filters. Our empirical analysis shows that the particle routing step enables the distribution of particles among sub-filters, and the re-sampling step can be effectively parallelized. Section. VIII-C shows that there is no substantial variation in the estimation error between the proposed modified SIR algorithm and the conventional algorithm. An algorithmic flowchart is shown in Fig. 2

The particles and their associated weights in sub-filter k at time step t are represented by $\{x_t^{(k,i)}, w_t^{(k,i)}\}_{i=1}^{M}$, for $k = 1, \cdots K$. The particle $x_t^{(k,i)}$ represents the position in the Cartesian co-ordinate system.

VII. ARCHITECTURE OVERVIEW

In this section, we present a high-speed architecture for PFs, based on the modified SIRF algorithm presented in Section VI.

The top-level architecture shown in Fig. 3 utilizes a filter bank consisting of K sub-filters working in parallel. Sampling, importance, and re-sampling operations are carried out within a sub-filter. In addition to the SIR step, a fixed number of particles are routed between sub-filters after the completion of every iteration as part of a particle routing operation. The sub-filters are connected based on ring-topology inside the filter bank. M particles are time-multiplexed and processed within each sub-filter, and $Q = M/2$ particles are exchanged with neighbouring sub-filters. Since the number of particles exchanged and the routing topology are fixed, the proposed architecture has very low design complexity. The design can be easily scaled up to process a large number of particles (N) by replicating sub-filters. The binary measurements of the eight photodiodes (z_t) are fed as an input to the filter bank along with the true bearing (ϕ_t^{UGV}) and the position of the UGV (x_t^{UGV}). Random number generation needed for the sampling and re-sampling steps is provided by a random number generator block. We use a parallel multiple output LFSR architecture presented by Milovanović et al. [33] for random number generation. A 16 bit LFSR is used since our internal variables are 16 bits wide. Further, a detailed description of the sub-filter architecture is provided in Section VII-A. The sector check block, described in Section VII-B, computes the particle population in each of the eight sectors and outputs a sector index that has the maximum particle population. This information is used by the UGV to traverse in the direction of the source. The mean computational block used to calculate the global mean of all N particles from K sub-filters to estimate the source location (pos_t), is explained in Section VII-C.

A. SUB-FILTER ARCHITECTURE

The sub-filter is the main computational block responsible for particle generation, processing, and filtering. It consists of three main sub-modules, namely, sampling, importance and re-sampling, as shown in Fig. 4. The sampling and importance blocks are pipelined in operation. The re-sampling step cannot be pipelined with the former steps as it requires weight information of all particles. Thus, it is started after the completion of the importance step. Since sampling and importance stages are pipelined, together they take M clock cycles to iterate for M particles, as shown in Algorithm 2 from line 5 to line 8. The particle routing between the sub-filters is done along with the sampling step and does not require any additional cycles. The re-sampling step takes $3M$ clock cycles, as discussed in Section VII-A3.

1) Sampling and routing

The sampling step involves generating new sampled particles $\{x_t^{(k,i)}\}_{i=1}^{M}$ by propagating re-sampled particles

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

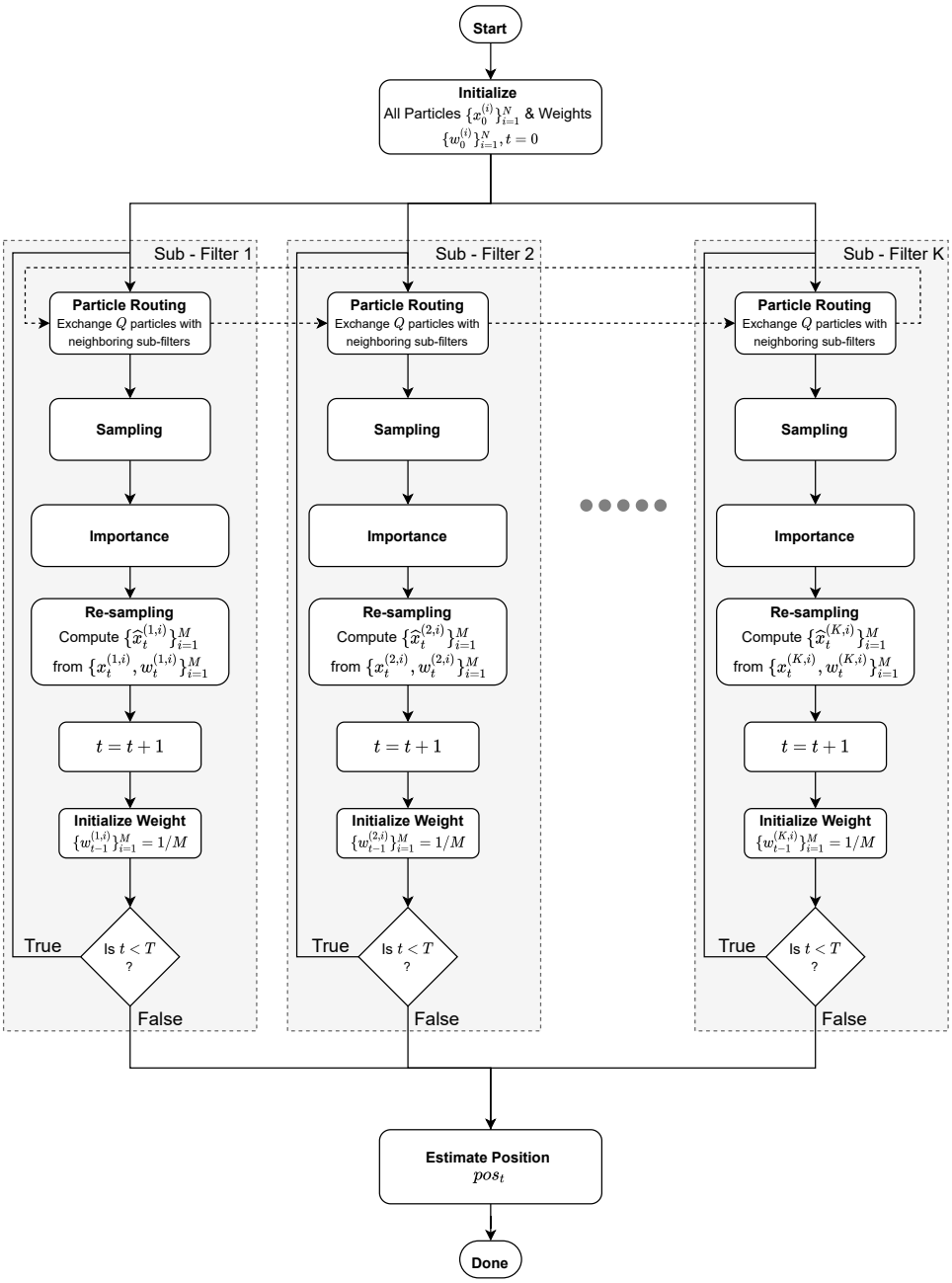


FIGURE 2: Flowchart illustrating the sequence of operations carried out incorporating the modified SIR algorithm. T represents the total time steps for localizing the source. [New figure]

$\{\widehat{x}_{t-1}^{(k,i)}\}_{i=1}^M$ from the previous time step using the dynamic state space model:

$$x_t^{(k,i)} \sim p(x_t|\widehat{x}_{t-1}^{(k,i)}) \tag{12}$$

Conventionally, particles $\{x_t^{(k,i)}\}_{i=1}^M$ are used to generate the weights $\{w_t^{(k,i)}\}_{i=1}^M$ in the importance unit, and using these weights we determine the re-sampled particles $\{\widehat{x}_t^{(k,i)}\}_{i=1}^M$. Further, $\{\widehat{x}_t^{(k,i)}\}_{i=1}^M$ is utilized to obtain particles $\{x_{t+1}^{(k,i)}\}_{i=1}^M$ of the next time step. Thus, with the straightforward approach, we would need two memories each of depth M to store $\{x_t^{(k,i)}\}_{i=1}^M$ and $\{\widehat{x}_t^{(k,i)}\}_{i=1}^M$ within a sub-filter. Similarly, for K sub-filters we would require $2 \times K$ memory elements, each of depth M. This increases memory usage for higher K or M. In this work, we suggest a novel scheme to store the particles using a single dual-port memory instead of two memory blocks, which brings down the total memory requirement for storing particles to K memory elements, each of depth M.

In this scheme, since the re-sampled particles are actually the subset of sampled particles (i.e.,$\{\widehat{x}_t^{(k,i)}\}_{i=1}^M \subset \{x_t^{(k,i)}\}_{i=1}^M$) instead of storing $\{\widehat{x}_t^{(k,i)}\}_{i=1}^M$ in a different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

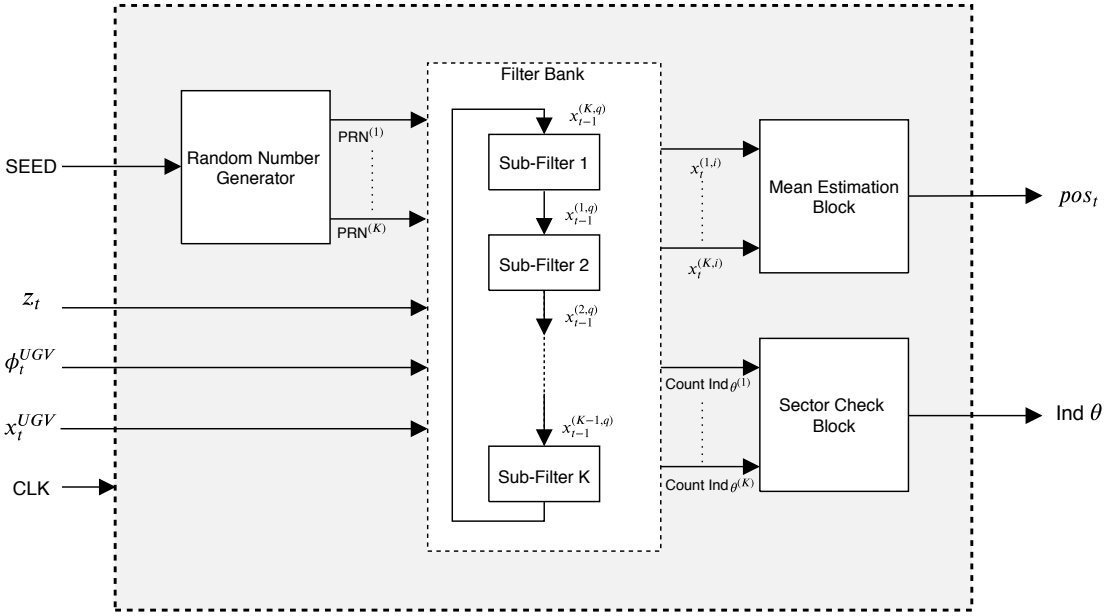


FIGURE 3: Top-level architecture of the realized particle filter algorithm.

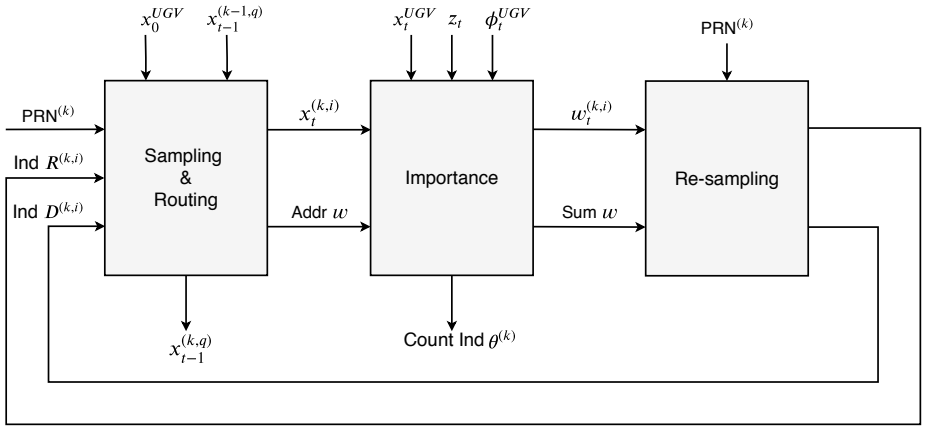


FIGURE 4: Sub-filter architecture.

memory, we can use the same memory as $\{x_t^{(k,i)}\}_{i=1}^{M}$ and use suitable pointers or indices to read $\{\widehat{x}_t^{(k,i)}\}_{i=1}^{M}$.

The re-sampling unit in our case is modified such that instead of returning re-sampled particles $\widehat{x}_{t-1}^{(k,i)}$, it returns the indices of replicated (Ind $R^{(k,i)}$) and discarded (Ind $D^{(k,i)}$) particles (cf. Fig. 4). Ind $R^{(k,i)}$ is used as a read address of the dual-port particle memory shown in Fig. 5 to point to the re-sampled particles $\widehat{x}_{t-1}^{(k,i)}$. The dual-port memory enables us to perform read and write operations simultaneously; however, this might result in data overwriting. For example, consider six particles, after re-sampling particle 2 ($x_{t-1}^{(k,2)}$) is replicated four times; particle 5 ($x_{t-1}^{(k,5)}$) is replicated two times and particles $1, 3, 4\,\&\,6$ are discarded. The re-sampling unit returns Ind $R = (2, 2, 2, 2, 5, 5)$ and Ind $D = (1, 3, 4, 6)$. The read sequence of the dual-port memory is $(2, 2, 2, 2, 5\ \&\ 5)$ and the write sequence is $(2, 1, 3, 4, 5\ \&\ 6)$. Initially, particle 2 ($x_{t-1}^{(k,2)}$) is read from the dual-port memory and after propagation in the sampling block, the sampled particle $x_t^{(k,i)}$ is written back to the memory location 2. Next, particle 2 is read again from memory location 2. However, this time the content of the location is changed, and it no longer holds the original particle $x_{t-1}^{(k,2)}$, which causes an error while reading. In order to avoid this scenario, we introduce a sub-block (a) (cf. Fig. 5), wherein when we read the particle from the memory for the first time, it is temporarily stored in a register. Hence, whenever there is a replication in Ind R or read address, we read the particle from the register instead of memory. The Rep signal is generated by comparing Ind R with its previous value and if both are same, Rep will be made high.

Further, we introduce a sub-block (b) (cf. Fig. 5), which is responsible for routing the particles between neighbouring

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

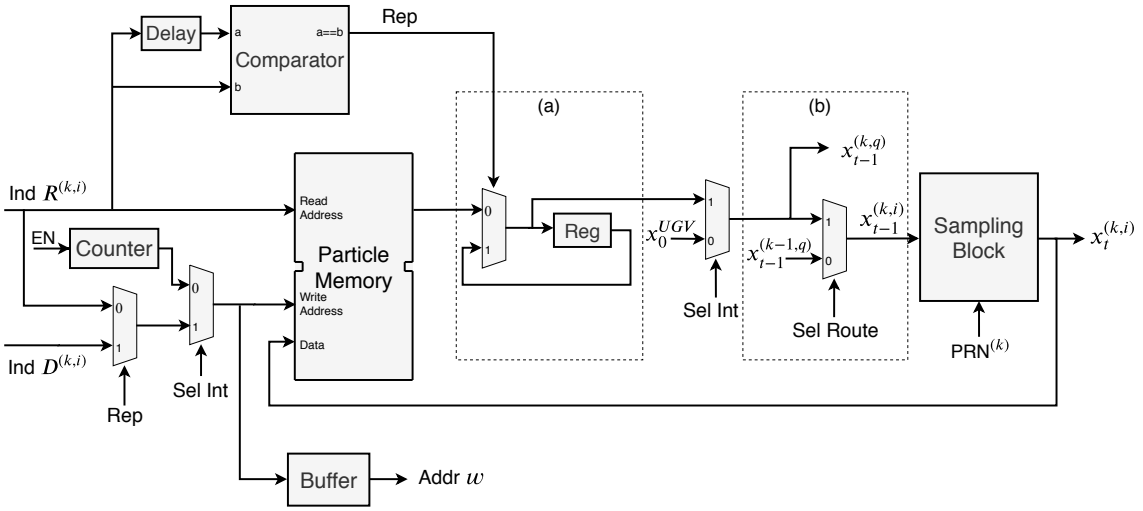


FIGURE 5: Sampling and routing unit architecture.

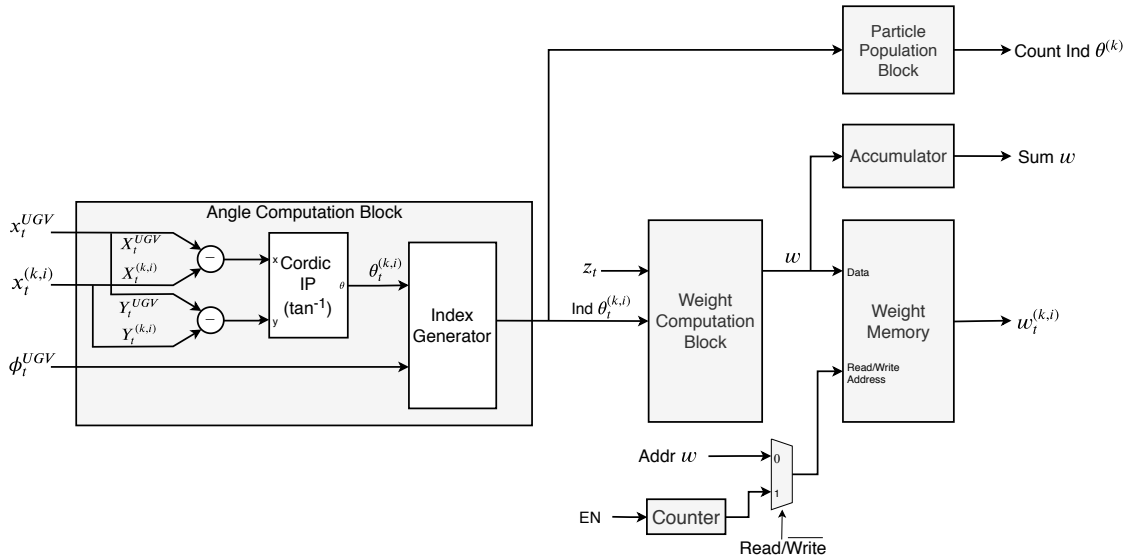


FIGURE 6: Importance unit architecture.

sub-filters. Out of M particles read from the particle memory of sub-filter k, the first $M/2$ particles, i.e., $\{x_{t-1}^{(k,q)}\}_{q=1}^{M/2}$ are sent to sub-filter $k+1$, and simultaneously the first $M/2$ particles, i.e., $\{x_{t-1}^{(k-1,q)}\}_{q=1}^{M/2}$, of sub-filter $k-1$ are read and fed to the sampling block of sub-filter k. The sampling block propagates the particles from time step $t-1$ to time step t. The routed particles from sub-filter $k-1$ $\{x_{t-1}^{(k-1,q)}\}_{q=1}^{M/2}$, and last $M/2$ local particles $\{x_{t-1}^{(k,q)}\}_{q=M/2+1}^{M}$ read from particle memory of sub-filter k are propagated by the sampling block and written back to the memory. The input to the sampling block are particles of time step $t-1$ $(x_{t-1}^{(k,i)})$ and the output are particles of current time step t $(x_t^{(k,i)})$. The sampling block pseudocode is provided by Algorithm 3. The random number $PRN^{(K)}$ needed for random sampling of particles as shown in Algorithm 3, line 2 and line 3 is provided by a random number generator block (cf. Fig. 3). The $Sel\,Route$ signal is used to control the switching between the local and routed particles by making it low for the first $M/2$ cycles and then making it high for the next $M/2$ cycles. Further, at time instant 0, we feed the UGV position x_0^{UGV} as a prior to the sampling block to distribute the particles around the UGV. The $Sel\,Int$ control signal is made low in the first iteration, i.e., at time instant 0, and then made high for the subsequent iterations.

2) Importance

The importance unit computes the weights of the particles based on the photodiode measurements z_t given by:

$$w_t^{(k,i)} = w_{t-1}^{(k,i)} p(z_t | x_t^{(k,i)}) \tag{13}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

Algorithm 3 : Sampling block pseudocode

Input: Particles from previous time step $x_{t-1}^{(k,i)} = [X_{t-1}^{(k,i)}, Y_{t-1}^{(k,i)}]$ and random number $PRN^{(k)} = [PRN_x^{(k)}, PRN_y^{(k)}]$.
Output: Particles of current time step $x_t^{(k,i)}$.
Method:

1: for $i = 1$ to M do
2: $X_t^{(k,i)} = X_{t-1}^{(k,i)} + PRN_x^{(k)} * std$
3: $Y_t^{(k,i)} = Y_{t-1}^{(k,i)} + PRN_y^{(k)} * std$ ▷ std is the standard deviation.
4: $x_t^{(k,i)} = [X_t^{(k,i)}, Y_t^{(k,i)}]$
5: end for

$w_{t-1}^{(k,i)}$ is initialized to 1/M. Estimation of $p(z_t|x_t^{(k,i)})$ involves determining the angle of each particle $(\theta_t^{(k,i)})$, which is computed using an inverse tangent function based on the position of the UGV (x_t^{UGV}) and position of the particle $(x_t^{(k,i)})$, as follows:

$$\theta_t^{(k,i)} = tan^{-1}\left(\frac{Y_t^{(k,i)} - Y_t^{UGV}}{X_t^{(k,i)} - X_t^{UGV}}\right)$$

where, $X_t^{(k,i)}$ and $Y_t^{(k,i)}$ represents the co-ordinates of the particle $x_t^{(k,i)}$ in two-dimensional Cartesian co-ordinate system.

The inverse tangent function is implemented using a Cordic IP block provided by Xilinx [34]. The architecture of the importance unit is shown in Fig. 6. The index generator block estimates the angle of the particles with respect to the longitudinal axis of the UGV based on the bearing of the UGV (ϕ_t^{UGV}). In addition to this, the index generator block is used for determining the sector indices (Ind $\theta_t^{(k,i)}$) of the particles based on the angle information. The sector indices of the particle can be defined as follows:

$$Ind\ \theta_t^{(k,i)} = \lceil 4/\pi * (\theta_t^{(k,i)} - \phi_t^{UGV}) \rceil$$

z_t is 8 bit wide data consisting of 8 binary photodiode measurements $\{z_t^1, z_t^2 \cdots z_t^8\}$. Based on the measurement z_t and the sector indices of particles, weights are generated by the weight computation block. These weights are stored in the weight memory using the address provided by the sampling unit, to store weights in the same order as the sampled particles $x_t^{(k,i)}$. The sum of all the weights required by the re-sampling unit is obtained by an accumulator. The particle population block is used to estimate the number of particles present in each of the eight sectors, using the sector indices of particles for a given sub-filter. The particle count in each of the eight sectors of sub-filter k is concatenated and given as the output Count Ind $\theta^{(k)}$. For example, if sector 1 has 15 particles, sector 3 has 14 particles, and sector 5 has 3 particles, then Count Ind $\theta^{(k)} = \{15, 0, 14, 0, 3, 0, 0, 0\}$.

3) Re-sampling

Particles with higher weights are replicated, while particles with lower weights are discarded during the re-sampling process. This is accomplished by utilizing a Systematic re-sampling algorithm shown in Algorithm 4. A detailed description of the systematic re-sampling algorithm is provided in [8], [28]. The weights and sum of all weights are obtained from the importance unit. The random number (U_0) needed to compute the parameter U_scale in line 2 of Algorithm 4 is provided by the Random number generator block shown in Fig. 3. The algorithm presented works with un-normalized weights, which will avoid M division operations on all particles to implement normalization. The division required to compute A_w in line 1 of Algorithm 4 is implemented using the right shift operation. This approach consumes fewer resources and area on hardware. The replicated and discarded indices generated by the systematic re-sampling block are stored in their respective memories, as shown in Fig. 7. In the worst-case scenario, the inner loop of Algorithm 4 takes $2M$ cycles for execution in hardware as it involves fetching M weights from weight memory and doing M comparison operations. Further, line 13 and line 14 take M cycles to obtain M replicated indices. Thus, in total, the execution of the re-sampling step requires $2M + M = 3M$ cycles.

Algorithm 4 : Systematic Re-sampling.

Input: Un-normalized weights $(\{w_t^{(k,i)}\}_{i=1}^{M})$ of M particles, summation of all the weights in a sub-filter (Sum w) and the uniform random number (U_0) between $[0, 1]$
Output: Replicated index (Ind R) and Discarded index (Ind D).
Method:

1: Compute $A_w = \frac{Sum\,w}{M}$
2: Initialize : $U\_scale = U_0 \times Aw$
3: $s = 0, p = 0, m = 0$
4: **for** i=1 to M **do**
5: **while** $s < U\_scale$ **do**
6: $p = p + 1$
7: $s = s + w^{(k,p)}$
8: **if** $s < U\_scale$ **then**
9: $m = m + 1$
10: Ind $D^{(k,m)} = p$
11: **end if**
12: **end while**
13: $U\_scale = U\_scale + A_w$
14: Ind $R^{(k,i)} = p$
15: **end for**

B. SECTOR CHECK BLOCK

The direction/orientation of the UGV is decided by the population of particles in different sectors and is used to move towards the source. This is achieved by the sector

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

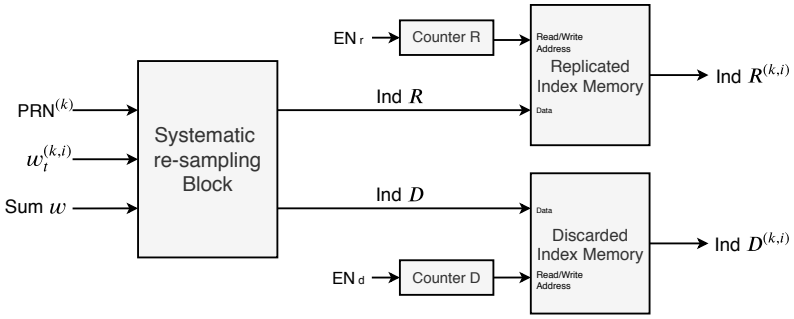


FIGURE 7: Re-sampling unit architecture.

check block, which estimates the particle population in each of the eight sectors and gives the sector index with maximum particle count. The block diagram shown in Fig. 8 utilizes eight parallel adders to count the number of particles in each sector. The particle count in a given sector of K sub-filters is fed as an input to the adder. Count Ind $\theta_n^{(k)}$ in Fig. 8 denotes the particle count in sector n of sub-filter k. The output of an adder gives the total particle population in a particular sector. Furthermore, the sector index (Ind θ) having the maximum particle count is estimated using a max computation block. The UGV uses this information to traverse in the direction of the source.

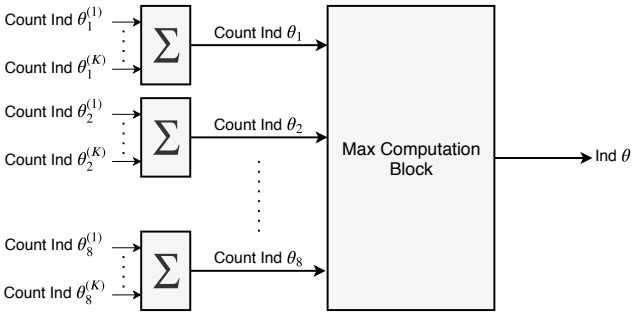


FIGURE 8: Sector check block architecture.

C. MEAN ESTIMATION BLOCK

The mean of total N particle positions is estimated using the mean estimation block. Particle positions from K sub-filters are fed in parallel and accumulated over M cycles to generate the sum, which is further divided by N, by right shifting $log_2(N)$ times to get the mean. In our implementation, we consider N as a power of 2. The mean gives an estimate of the position of the source pos_t.

VIII. RESULTS

In this section, we present the resource utilization of the proposed design on an FPGA. We also evaluate the execution time of the proposed architecture as a function of the number of sub-filters and inspect the estimation accuracy by scaling the number of particles. We then compare our design to the current state-of-the-art implementations. Furthermore, we present experimental results for the source localization problem implemented on FPGA using the proposed architecture.

A. RESOURCE UTILIZATION

The architecture presented was implemented on Artix-7 FPGA. Resource utilization of the implemented design for the different number of sub-filters is summarized in Table 2. The number of particles per sub-filter (M) was fixed to 32 for synthesizing the design. All memory modules shown in the architecture for storing particles, weights, replicated, and discarded indices are translated into embedded 18kb block random access memory (BRAM) available on the FPGA, using a block memory generator (BMG) IP [35] provided by Xilinx. The number of 18kb BRAM blocks needed on the FPGA is indicated in the Block RAM column of Table 2. It can be seen that the resource utilization increases proportionally with the number of sub-filters. For 64 sub-filters, 64% of the slice LUTs (lookup tables) are used, and a maximum of approximately 90 sub-filters can fit onto a single Artix-7 (xc7a200tfbg484-1) FPGA platform.

B. EXECUTION TIME

The proposed design utilizes K parallel sub-filters, thus bringing down the number of particles processed within a sub-filter to N/K. Since, sampling and importance blocks are pipelined, these steps take $N/K + \tau_s + \tau_i$ clock cycles and the re-sampling step takes $3N/K + \tau_r$ cycles to process N/K particles, where τ_s , τ_i and τ_r represent the start-up latency of the sampling, importance and re-sampling units, respectively. Since all the K sub-filters are parallelized, the time taken to process a total of N particles for SIR operation is:

$$T_{SIR} = (4N/K + \tau)T_{clk}$$

where, $\tau = \tau_s + \tau_i + \tau_r$ and T_{clk} is the clock period of the design.

Fig. 9 gives the timing diagram for completion of SIR operations using the proposed architecture for N particles, for a single iteration. Furthermore, since particle routing is incorporated within the sampling step, the transfer of particles between the sub-filters do not take any additional

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

TABLE 2: Resource utilization on Artix-7 FPGA.

Sub-filters K	Occupied slices	Slice LUTs	LUTRAM	Slice Registers	Block RAM
1	505 (1.5%)	1,437 (1.07%)	51 (0.11%)	1,735 (0.65%)	2 (0.55%)
2	942 (2.8%)	2,847 (2.13%)	102 (0.22%)	3,259 (1.22%)	4 (1.10%)
4	1,710 (5.1%)	5,494 (4.11%)	204 (0.44%)	6,297 (2.35%)	8 (2.19%)
8	3,465 (10.3%)	10,973 (8.20%)	408 (0.88%)	12,356 (4.62%)	16 (4.38%)
16	6,834 (20.3%)	21,885 (16.36%)	816 (1.77%)	24,460 (9.14%)	32 (8.77%)
32	14,513 (43.1%)	43,804 (32.74%)	1,632 (3.53%)	48,645 (18.18%)	64 (17.53%)
64	29,290 (87%)	86,101 (64.35%)	3,264 (7.06%)	95,570 (35.71%)	128 (35.07%)

cycles. This makes the design scalable for a large number of sub-filters, as the routing operation requires no extra time.

In Fig. 10(a), we show the execution time of the proposed architecture as a function of the number of sub-filters (K) for different N. As expected, the execution time increases with the number of particles (N). In many applications, for example, in biomedical signal processing, the state space dimension is very high [7]. Consequently, a large number of particles are needed to obtain satisfactory performance. In such cases, the computation time often becomes unrealistic. Introducing parallelization in the design by using more sub-filters (K) brings down the execution time significantly, as shown in Fig. 10(a). However, the reduction in execution time by increasing K comes at the cost of added hardware, which can be inferred from Fig. 10(b). Thus, there is a trade-off between the speed and the hardware utilized. For instance, using a single sub-filter and no parallelization uses a mere 1.4k (1%) LUTs to process 256 particles, and the time taken for SIR operations is around 1075 clock cycles. On the other hand, an 8 sub-filter design takes only 178 clock cycles for SIR operations, but utilizes 11k (8%) LUTs. Thus, there is a trade-off between speed and hardware used. The given FPGA resources limit the total number of sub-filters that can be accommodated on an FPGA, thus limiting the maximum achievable speed.

C. ESTIMATION ACCURACY

We analyzed the estimation accuracy for the 2D source localization problem as a function of the number of particles (N) for the standard and the modified SIR algorithm. The estimation error gives the error between the actual source location and the estimated source location given by:

$$Error = \sqrt{(pos_x - x)^2 + (pos_y - y)^2} \qquad (14)$$

where, pos_x and pos_y denote the estimated position of the source obtained from the PF algorithm, in the 2D Cartesian co-ordinate system. x and y denote the true position of the source in the $2D$ arena.

The algorithm for the standard SIR filter is presented in Section. IV and has no parallelization incorporated. The modified SIR algorithm implements parallelization by utilizing K sub-filters working concurrently to reduce the execution time, introduced in Section. VI. The estimation errors presented in Fig. 10(c) are the average errors in 1000 runs over 250 time-steps. It is inferred that there is no significant difference in the estimation error between the standard and the modified SIR algorithm. Additionally, the modified algorithm achieves lower execution time and allows the parallel computation of PFs. Further, it is noted that by scaling the number of particles, the estimation accuracy improves as the error decreases.

D. CHOICE OF THE NUMBER OF SUB-FILTERS K

Choice of the number of sub-filters (K) used in the design depends on several factors such as, the number of particles (N), the clock frequency of the design (f_{clk}), and the observation sampling rate (f_s) of the measurement samples. The sampling rate gives the rate at which new input measurements can be processed. N is chosen depending on the application for which the particle filter is applied. f_{clk} is selected based on the maximum frequency supported by the design. The relationship between the sampling rate and the execution time (T_{SIR}) of the filter is given by:

$$f_s = 1/T_{SIR} = \frac{f_{clk}}{(4N/K + \tau)}$$

where, $f_{clk} = 1/T_{clk}$. Thus, for a specified measurement sampling rate (f_s), the clock frequency of the design (f_{clk}), and the number of particles (N), we can determine the number of sub-filters (K) needed from the above equation. For instance, in our application, we use 256 particles because the error curve levels off at $N = 256$ (cf. Fig. 10(c)), and there is no improvement in the estimation error by further increasing N. Thus, to achieve a sampling rate of $f_s = 562$ kHz, with 256 particles and clock frequency $f_{clk} = 100$ MHz, we utilize $K = 8$ sub-filters. The maximum number of sub-filters that can be used in the design depends on the resources of the given FPGA.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

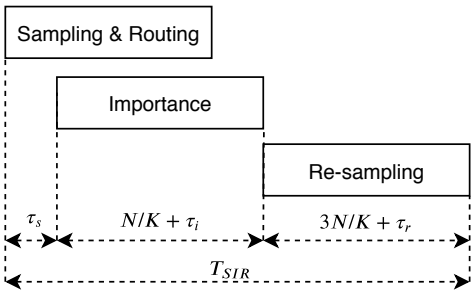


FIGURE 9: Timing diagram for SIR operations of the proposed design.

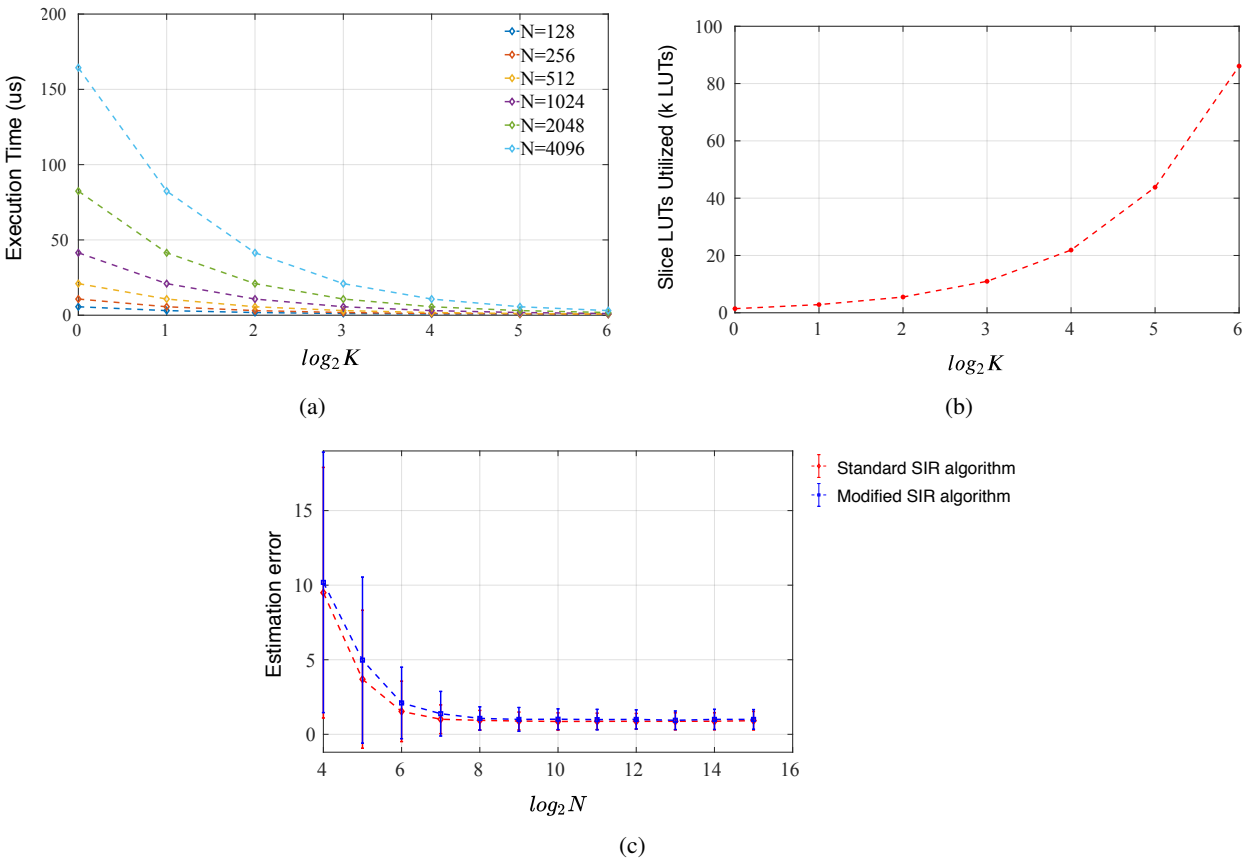


FIGURE 10: Performance analysis of the proposed design. (a) Execution time of the proposed design as a function of the number of sub-filters (K), for different number of particles (N). (b) Resource utilization in terms of the number of slice LUTs used as a function of the number of sub-filters (K). (c) Estimation error as a function of the number of particles (N) for the standard SIR filter without any parallelization using Algorithm 1 and the modified SIR filter with parallelization using Algorithm 2.

E. COMPARISON WITH STATE-OF-THE-ART IMPLEMENTATIONS

A comparison of our design with state-of-the-art implementations is provided in Table 3. To obtain a valid assessment with other works, we have used $N = 1,024$ particles (although 256 particles are sufficient for our application as error curve levels off at $N = 256$ (cf. Fig. 10(c)) and $K = 8$ sub-filters for comparison. The majority of current implementation schemes use the standard SIR algorithm (cf. Algorithm 1), which does not support parallelization. Moreover, their architectures are not scalable to process a large number of particles at the high sampling rate, as the execution time is proportional to the number of particles. Also, the re-sampling step is a major computational bottleneck, as it is inherently not parallelizable. In this work, we propose a modification to the existing algorithm that overcomes this computational bottleneck of the PF algorithm and makes the high-speed implementation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

TABLE 3: Performance summary and comparison with state-of-the-art particle filter implementation schemes. [Table Updated]

Reference	Application	Device	Number of particles (N)	Resources	Execution Time (μs) with 100 MHz clock	Sampling Rate (kHz)
Athalye et al. [8]	Tracking	Virtex II pro	2048	4.39k FFs, 3.85k LUTs, 18 BRAMs & 13 DSPs	60.24	16
Agarwal et al. [10]	Object tracking in video	Virtex-5	31	59.39k LUTs, 52 BRAMs & 2 DSPs	23.5	42.55
Ye and Zhang [11]	Tracking	Virtex-5	1024	13.69k FFs, 7.38 LUTs, 13 BRAMs & 4 DSPs	21.74	46
Sileshi et al. [12]	Localization	Kintex-7	1024	1.46k FFs, 19.12k LUTs, 86 DSPs & 260 BRAMs	55.37	18
Velmurugan [15]	Tracking	Virtex II pro	1000	17.42k FFs, 30.9k LUTs & 3 DSPs	33.33	30
Schwiegelshohn et al. [16]	Position estimation	Zynq-7020	14	5.93k FFs, 52.41k LUTs & 214 DSPs	6†	166.67†
Mountney et al. [17]	Brain Machine Interfaces	Not Provided	1000	Not Provided	~ 11	~ 90
Alam et al. [18]	Generic	Virtex-6	1000	299‡ LUTs & 2‡ BRAMs (For re-sampling)	Not Provided	
Miao et al. [19]	Tracking of Multiple Sources of Neural Activity	Virtex-5	3200	43.64k FFs, 42.38k LUTs, 134 BRAMs & 283 DSPs	48.52	20.61
This Work	Localization	Artix-7	1024	12.35k FFs, 10.97k LUTs & 16 BRAMs	**5.62***	**178***

* $K = 8$ sub-filters are used for the calculation.
† 14 real particles are used for the evaluation [16].
‡ On hardware, only the re-sampling stage is implemented.

possible. We introduce an additional particle routing step (cf. Algorithm 2) allowing for parallel re-sampling. We develop a PF architecture based on the modified algorithm incorporating parallelization and pipelining design strategies to reduce the execution time. Since the particle routing step is coupled with the sampling step and the routing is constrained between the two neighboring sub-filters, our implementation is highly scalable and has low complexity. In comparison, other parallel implementations suffer from scalability issues due to the high communication overhead between the concurrent processing elements.

Despite the difficulty of directly comparing the proposed architecture to other implementations owing to variation in model, application, device, and particle count (N), our design achieves high input sampling rates, even for a large number of particles, by scaling the number of sub-filters K. The first hardware architecture for implementing PFs on an FPGA was provided by Athalye et al. [8], applied to a tracking problem. Their architecture is generic and does not incorporate any parallelization in the design. Thus, their architecture suffers from a low sampling rate of about 16 kHz for 2048 particles, which is approximately 11 times lower than the sampling rate of our design. However, owing to non-parallel architecture, the resource consumption of their design (4.4k registers and 3.8k LUTs) is relatively low. Agarwal et al. [10] proposed a PF architecture for object tracking in video with 59k LUTs and a sampling rate of around 42kHz. Another state-of-the-art system was presented in [11]. The authors implemented the SIR filter on the Xilinx Virtex-5 FPGA platform for bearings-only tracking application and achieved a sampling rate of 46 kHz for 1024 particles. Regarding its hardware utilization, it uses 13.6k registers and 7.3k LUTs, which are comparable to those of our design; however, their sampling rate is four times lower than that of our system. Sileshi et al. [12] proposed two methods for implementing PFs on hardware. The first method was a hardware/software (HW/SW) co-design framework, where the software components were implemented using an embedded MicroBlaze processor. A PF hardware acceleration module on an FPGA was used for the hardware portion. This HW/SW co-design approach has a low sampling rate of about 1 kHz due to communication overhead between the MicroBlaze soft processor and the hardware acceleration module. Furthermore, using a large number of parallel particle processors to speed up the design is constrained by the number of bus interfaces available in the soft-core processor (MicroBlaze). Thus, to improve the sampling rate, they proposed a second approach which is entirely a hardware design. However, their architecture does not support parallel processing and achieves a low sampling rate of about 18 kHz, whereas our system can sample at 178 kHz for processing the same 1024 par-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

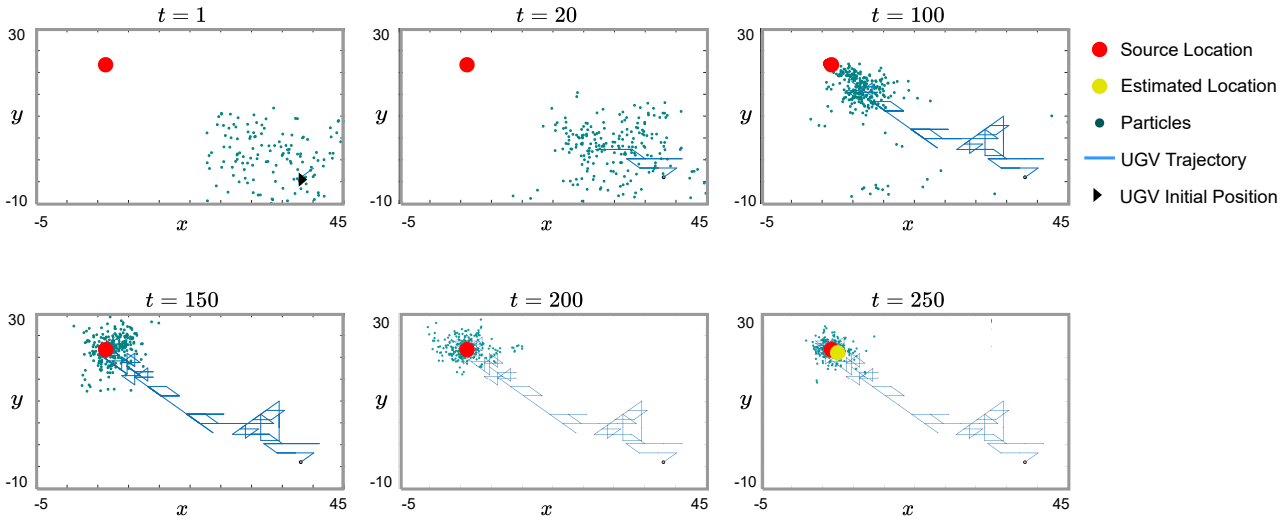


FIGURE 11: 2D source localization experimental result. The source is positioned at $[6, 22]$ marked by a 'red' circular dot. At the start, the UGV is positioned at $[38, -4]$. The model is run over 250 time-steps for 256 particles, and the UGV traverses towards the source based on sensor measurements. The final source estimate (pos_t) obtained by the PF algorithm is marked by a 'yellow' circular dot and has an estimation error of 0.5. The probabilities α and β are set as 0.8 and 0.6, respectively. [Figure Updated]

ticles. Their full hardware system utilizes 1.4k registers and 19k LUTs. Velmurugan [15] proposed a fully digital PF FPGA implementation for tracking application, without any parallelization in the design. They used a high-level Xilinx system generator tool to generate the VHDL code for deployment on a Xilinx FPGA from Simulink models or MATLAB code. Their design is not optimized in terms of hardware utilization as they use a high-level abstraction tool and lack flexibility to fine-tune the design. On the other hand, our design is completely hand-coded in Verilog and provides granular control to tweak the design parameters, and ensures that the design can be easily integrated into a multitude of PF applications. They achieve a sampling rate of about 30 kHz for 1000 particles, which is six times lower than that of our design. Further, their resource consumption is relatively high (17.4k registers and 30.9k LUTs) as they use high-level abstraction tools for implementation. In other work, Schwiegelshohn et al. [16] proposed an FPGA optimized re-sampling to support parallelism and demonstrated the hardware implementation for meager 14 particles with 5.93k registers and 52.41k LUTs. Due to low particle count, they achieve a sampling rate of around 166 kHz. Mountney et al. [17] implemented a Bayesian auxiliary particle filter algorithm (BAPF) on an FPGA for brain machine interfaces with a 90 kHz sampling rate. Alam et al. [18] proposed an improved multinomial re-sampling scheme to reduce the re-sampling latency and implemented the same on an FPGA with 299 LUTs and 2 BRAMs for 1k particles. However, they don't report the sampling rate of the whole architecture. Miao et al. [19] introduced a probability hypothesis density filtering for tracking multiple sources of neural activity. The sampling and weight update steps are distributed over concurrent PEs, and the re-sampling is done within a CU. The communication cost between the multiple PEs and CU increases linearly with the number of PEs, making the architecture not scalable to process a large number of particles. In contrast, the re-sampling in our architecture is local to the sub-filter and the ring topology employed limits the communication to adjacent sub-filters, thereby reducing the routing overhead. Their implementation utilizes 43.6k flip flops (FFs) and 42.3k LUTs with sampling rate of 20 kHz for 3200 particles.

Our system has a comparable resource utilization (12.3k registers and 10.9k LUTs for 8 sub-filters) with a low execution time of about 5.62 μs and achieves a sampling rate of about 178 kHz. Our design can be used in real-time applications due to the low execution time. Further, to achieve a high sampling rate even with a large number of particles, more sub-filters can be used, as shown in Fig. 10(a). However, this comes at the expense of additional hardware. On the other hand, the resource utilization of our system can go as low as 1.7k registers and 1.4k LUTs using a single sub-filter (cf. Table 2) for applications that have stringent resource constraints, but at the cost of increased execution time (cf. Fig. 10(a)).

F. EXPERIMENTAL RESULTS

The design was implemented on an Opal Kelly board [36] with Artix-7 FPGA for 2D source localization. The implementation result is shown in Fig. 11. The state in this 2D

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

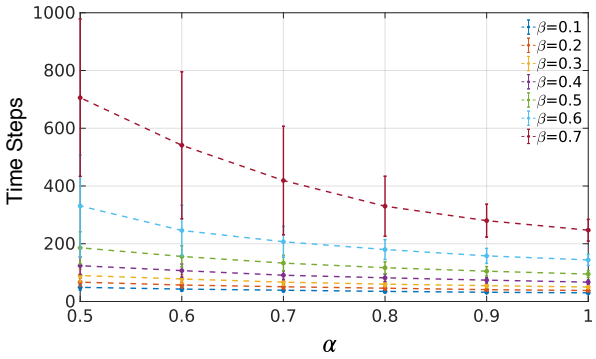


FIGURE 12: Variation in the number of time-steps required to localize the source as a function of α and β.

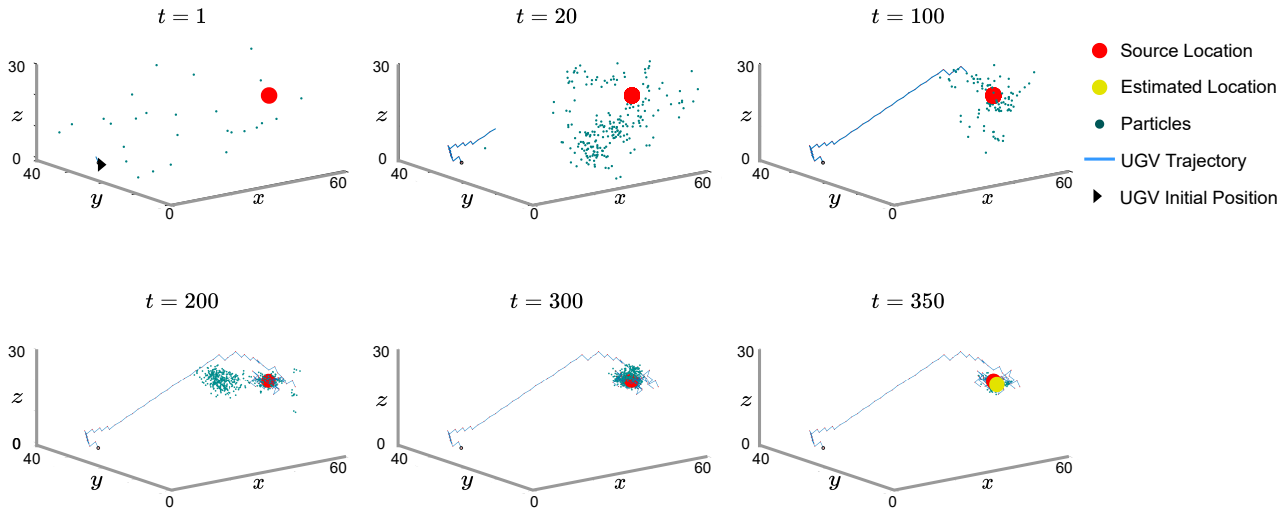


FIGURE 13: 3D source localization experimental result. The source is positioned at $[40, 5, 25]$, and the initial position of the UGV is $[10, 30, 0]$. The model is run over 350 time-steps for 512 particles. Here, the UAV traverses in three dimensions to move towards the source. The error between the source and the estimated location is 0.83. The probabilities α and β are set as 0.8 and 0.4, respectively. [Figure updated]

model is 2-dimensional and incorporates position in x and y directions. The input to the design is binary measurements from a set of 8 photodiodes and the instantaneous position of the UGV. Inputs are sampled and processed by the PF system over 250 time-steps on an FPGA to estimate the source location. We consider the probabilities α and β to be 0.8 and 0.6, respectively. It can be seen that the algorithm is robust enough to localize the source even with a noise probability of 0.6. However, with an increase in noise probability (β), the number of time-steps or iterations required to localize the source also increases, as shown in Fig. 12. The source is considered to be localized if the estimation error is less than the predetermined threshold, which is 2.5 in our case. The time-steps shown in Fig. 12 are the average time required to localize the source over 500 runs. The entire design was coded in Verilog HDL, and the design was implemented on FPGA.

All variables were translated from the floating-point to the fixed-point representation for the implementation on FPGA. We have used a 16-bit fixed-point representation for particles and their associated weights. All bearing-related information, such as the angle of the UGV and the angle of particles used in the importance block, is represented by a 12-bit fixed-point representation. Further, the indices of the replicated and the discarded particles are integers and are represented using $log_2(M) = 5$ bits. The output estimate of the source location (pos_t) is represented using a 16-bit representation. $N = 256$ particles were used for processing. $K = 8$ sub-filters were used in the design with $M = 32$ particles processed within each sub-filter. $M/2 = 16$ particles were exchanged between the sub-filters after the completion of every iteration as part of the particle routing operation. The time taken to complete SIR operations for $N = 256$ and $K = 8$ is 178 clock

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

cycles. With a clock frequency of 100 MHz, the speed at which we can process new samples is around 562 kHz, and the execution time for SIR operation is 1.78 μs. This high sampling rate enables us to use the proposed hardware architecture in various real-time applications.

Further, we show that the 2D source localization problem can be extended to 3D, and we have modelled it in software using MATLAB. This 3D model incorporates position along the x, y, and z directions. Here, an Unmanned Aerial Vehicle (UAV) can be utilized to localize the source. As compared to 8 sensors used in 2D localization, here we utilize 16 sensors for scanning the entire 3D space. We consider $\alpha = 0.8$ and $\beta = 0.4$, and the model was run over 350 time-steps for 512 particles to localize the source. The result is presented in Fig. 13. The estimation error between the actual source location and the estimated source location in the 3D arena is given by:

$$Error = \sqrt{(pos_x - x)^2 + (pos_y - y)^2 + (pos_z - z)^2} \tag{15}$$

where, pos_x, pos_y and pos_z denote the estimated position of the source obtained from PF algorithm, in the 3D Cartesian co-ordinate system. x, y and z represent the true position of the source in the $3D$ arena.

IX. CONCLUSIONS AND OUTLOOK

In this paper, we presented an architecture for the hardware realization of PFs, particularly sampling, importance, and re-sampling filters, on an FPGA. PFs perform better than traditional Kalman filters in non-linear and non-Gaussian settings. Interesting insights into the advantages of PFs, performance comparison, and trade-offs of PFs over other non-PF solutions are provided by [37], [38]. However, PFs are computationally very demanding and take a significant amount of time to process a large number of particles; hence, PFs are seldom used for real-time applications. In our architecture, we try to address this issue by exploiting parallelization and pipelining design techniques to reduce the overall execution time, thus making the real-time implementation of PFs feasible. However, a major bottleneck in high-speed parallel implementation of the SIR filter is the re-sampling step, as it is inherently not parallelizable and cannot be pipelined with other operations. In this regard, we modified the standard SIR filter to make it parallelizable. The modified algorithm has an additional particle routing step and utilizes several sub-filters working concurrently and performing SIR operations independently on particles to reduce the overall execution time. Our implementation is highly scalable and has low complexity since the particle routing step is integrated with the sampling step, and the routing is confined between the two adjacent sub-filters. On the other hand, other parallel architectures have scalability issues due to the high communication overhead between the concurrent processing elements.

A performance assessment in terms of the resource utilized on an FPGA, execution time, and estimation accuracy is presented. We also compared the estimation error of the modified SIR algorithm with that of the standard SIR algorithm and noted that there is no significant difference in the estimation error. The proposed architecture has a total execution time of about 5.62 μs (i.e., a sampling rate of 178 kHz) by utilizing 8 sub-filters for processing $N = 1024$ particles. We compared our design with state-of-the-art FPGA implementation schemes and found that our design outperforms other implementation schemes in terms of execution time. The low execution time (i.e., high input sampling rate) makes our architecture ideal for real-time applications.

The proposed PF architecture is not limited to a particular application and can be used for other applications by modifying the importance block of the sub-filter. The sampling and re-sampling block designs are generic and can be used for any application.

We also present a novel source localization model to estimate the position of a source based on received sensor measurements. Our PF implementation is robust to noise and can predict the source position even with a high noise probability. Experimental results show the estimated source location with respect to the actual location for 2D and 3D settings and demonstrate the effectiveness of the proposed algorithm.

In recent times, there has been an increase in the utilization of UGVs in several instances, such as disaster relief, and military applications, due to reduced human involvement and the ability to carry out the task remotely. The proposed source localization model using PFs can autonomously navigate and localize the source of interest without any human intervention, which would be very helpful in missions wherein there is an imminent threat involved, such as locating chemical, biological or radiative sources in an unknown environment. Further, the proposed PF framework and its hardware realization would be useful for the signal processing community for solving various state estimation problems such as tracking, navigation, and positioning in real-time.

REFERENCES

[1] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, Apr. 1993.

[2] M. Tian, Y. Bo, Z. Chen, P. Wu, and C. Yue, "Multi-target tracking method based on improved firefly algorithm optimized particle filter," *Neurocomputing*, vol. 359, pp. 438 – 448, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231219308240

[3] N. Merlinge, K. Dahia, and H. Piet-Lahanier, "A Box Regularized Particle Filter for terrain navigation with highly non-linear measurements," *IFAC-Papers*, vol. 49, no. 17, pp. 361 – 366, 2016.

[4] Z. Zhang and J. Chen, "Fault detection and diagnosis based on particle filters combined with interactive multiple-model estimation in dynamic process systems," *ISA Transactions*, vol. 85, pp. 247 – 261, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S001905781830394X

[5] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*,

vol. 10, no. 3, pp. 197–208, July 2000. [Online]. Available: https://doi.org/10.1023/A:1008935410038

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

[7] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "Multiple sensor sequential tracking of neural activity: Algorithm and FPGA implementation," in *2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*, 2010, pp. 369–373.

[8] A. Athalye, M. Bolić, S. Hong, and P. M. Djurić, "Generic hardware architectures for sampling and resampling in particle filters," *EURASIP Journal on Advances in Signal Processing*, vol. 2005, no. 17, p. 476167, Oct. 2005. [Online]. Available: https://doi.org/10.1155/ASP.2005.2888

[9] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2442–2450, July 2005.

[10] S. Agrawal, P. Engineer, R. Velmurugan, and S. Patkar, "FPGA implementation of particle filter based object tracking in video," in *2012 International Symposium on Electronic System Design (ISED)*, Dec. 2012, pp. 82–86.

[11] B. Ye and Y. Zhang, "Improved FPGA implementation of particle filter for radar tracking applications," in *2009 2nd Asian-Pacific Conference on Synthetic Aperture Radar*, 2009, pp. 943–946.

[12] B. G. Sileshi, J. Oliver, and C. Ferrer, "Accelerating particle filter on FPGA," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 591–594.

[13] B. Sileshi, C. Ferrer, and J. Oliver, "Chapter 2 - Accelerating Techniques for Particle Filter Implementations on FPGA," in *Emerging Trends in Computational Biology, Bioinformatics, and Systems Biology*, 2015, pp. 19 – 37. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128025086000028

[14] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015303201

[15] R. Velmurugan, "Implementation strategies for particle filter based target tracking," *Ph.D thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology*, May 2007.

[16] F. Schwiegelshohn, E. Ossovski, and M. Hübner, "A Fully Parallel Particle Filter Architecture for FPGAs," in *Applied Reconfigurable Computing*, K. Sano, D. Soudris, M. Hübner, and P. C. Diniz, Eds. Cham: Springer International Publishing, 2015, pp. 91–102.

[17] J. Mountney, I. Obeid, and D. Silage, "Modular particle filtering FPGA hardware architecture for brain machine interfaces," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2011, pp. 4617–4620.

[18] S. A. Alam and O. Gustafsson, "Improved Particle Filter Resampling Architectures," *Journal of Signal Processing Systems*, vol. 92, no. 6, pp. 555–568, Jun 2020. [Online]. Available: https://doi.org/10.1007/s11265-019-01489-y

[19] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "Efficient Bayesian Tracking of Multiple Sources of Neural Activity: Algorithms and Real-Time FPGA Implementation," *IEEE Transactions on Signal Processing*, vol. 61, no. 3, pp. 633–647, 2013.

[20] R. Velmurugan, S. Subramanian, V. Cevher, D. Abramson, K. M. Odame, J. D. Gray, H.-J. Lo, J. H. McClellan, and D. V. Anderson, "On low-power analog implementation of particle filters for target tracking," in *2006 14th European Signal Processing Conference*, 2006, pp. 1–5.

[21] G. Hendeby, J. D. Hol, R. Karlsson, and F. Gustafsson, "A graphics processing unit implementation of the particle filter," in *2007 15th European Signal Processing Conference*, Sep. 2007, pp. 1639–1643.

[22] G. Hendeby, R. Karlsson, and F. Gustafsson (EURASIP Member), "Particle Filtering: The Need for Speed," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 1, p. 181403, Jun 2010. [Online]. Available: https://doi.org/10.1155/2010/181403

[23] M. Chitchian, A. Simonetto, A. S. van Amesfoort, and T. Keviczky, "Distributed computation particle filters on GPU architectures for real-time control applications," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2224–2238, Nov. 2013.

[24] P. Gong, Y. O. Basciftci, and F. Ozguner, "A parallel resampling algorithm for particle filtering on shared-memory architectures," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, pp. 1477–1483.

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

[26] S. Kim, J. Cho, and D. Park, "Moving-Target Position Estimation Using GPU-Based Particle Filter for IoT Sensing Applications," *Applied Sciences*, vol. 7, no. 11, 2017. [Online]. Available: https://www.mdpi.com/2076-3417/7/11/1152

[27] L. M. Murray, A. Lee, and P. E. Jacob, "Parallel resampling in the particle filter," *Journal of Computational and Graphical Statistics*, vol. 25, no. 3, pp. 789–805, 2016. [Online]. Available: https://doi.org/10.1080/10618600.2015.1062015

[28] M. Bolić, A. Athalye, P. M. Djurić, and S. Hong, "Algorithmic modification of Particle Filters for hardware implementation," in *2004 12th European Signal Processing Conference*, Sep. 2004, pp. 1641–1644.

[29] Y. Q. Zhang, T. Sathyan, M. Hedley, P. H. W. Leong, and A. Pasha, "Hardware efficient parallel particle filter for tracking in wireless networks," in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, Sept. 2012, pp. 1734–1739.

[30] P. B. Choppala, P. D. Teal, and M. R. Frean, "Particle filter parallelisation using random network based resampling," in *17th International Conference on Information Fusion (FUSION)*, 2014, pp. 1–8.

[31] C. S. Thakur, S. Afshar, R. M. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "Bayesian Estimation and Inference Using Stochastic Electronics," *Frontiers in Neuroscience*, vol. 10, p. 104, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00104

[32] J. L. Palmer, R. Cannizzaro, B. Ristic, T. Cheah, C. V. D. Nagahawatte, J. L. Gilbert, and S. Arulampalam, "Source localisation with a Bernoulli particle-filter-based bearings-only tracking algorithm," 2015.

[33] E. Milovanović, M. Stojcev, I. Milovanović, T. Nikolic, and Z. Stamenkovic, "Concurrent generation of pseudo random numbers with LFSR of Fibonacci and Galois type," *Computing and Informatics*, vol. 34, Aug. 2015.

[34] "CORDIC v6.0 LogiCORE IP Product Guide," *Xilinx, Product Guide*, pp. 1–66, 2017.

[35] "Block Memory Generator v8.4," *Xilinx, LogiCORE IP Product Guide*, pp. 1–129, 2017.

[36] "Opal Kelly XEM7310," (Date last accessed 10-May-2021). [Online]. Available: https://opalkelly.com/products/xem7310/

[37] F. Ababsa, M. Mallem, and D. Roussel, "Comparison between particle filter approach and Kalman filter-based technique for head tracking in augmented reality systems," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, 2004, pp. 1021–1026.

[38] N. Y. Ko and T. G. Kim, "Comparison of Kalman filter and particle filter used for localization of an underwater vehicle," in *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2012, pp. 350–352.

ADITHYA KRISHNA received the Bachelor of Engineering degree from the Department of Electronics and Communication Engineering, PES Institute of Technology, Bangalore, India, in 2017. He is currently working as a Research Assistant at NeuRonICS lab, Department of Electronics Systems Engineering, Indian Institute of Science (IISc), since 2018. His research interests lie in VLSI architecture design, embedded system design, neuromorphic computing, and machine learning. He is a recipient of the summer research fellowship 2016 under the Indian Academy of Sciences, Bangalore.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3094962, IEEE Access

ANDRÉ VAN SCHAIK is a pioneer in Neuromorphic Engineering and the director of the International Centre for Neuromorphic Systems at Western Sydney University. He received his M.Sc. degree in electrical engineering from the University of Twente, Enschede, The Netherlands, in 1990 and his PhD degree in neuromorphic engineering from the Swiss Federal Institute of Technology (EPFL) Lausanne, Switzerland, in 1998. From 1991 until 1994, he was a researcher at the Swiss Centre for Electronics and Microtechnology (CSEM), where he developed the first commercial neuromorphic chip – the optical motion detector used in Logitech trackballs. In 1998 he was a postdoctoral research fellow in the Department of Physiology at the University of Sydney, and in 1999 he became a Senior Lecturer in their School of Electrical and Information Engineering and a Reader in 2004. In 2011 he became a full Professor at Western Sydney University. His research focuses on all aspects of neuromorphic engineering, encompassing neurophysiology, computational neuroscience, software and algorithm development, and electronic hardware design. He is a Fellow of the IEEE for contributions to Neuromorphic Circuits and Systems. He has authored more than 200 papers and is an inventor of more than 35 patents. In addition, he has founded three technology start-ups.

CHETAN SINGH THAKUR is an Assistant Professor at Indian Institute of Science (IISc), Bangalore, and has an adjunct faculty appointment at the International Center for Neuromorphic Systems, Western Sydney University (WSU), Australia. He received his PhD in neuromorphic engineering at the MARCS Research Institute, Western Sydney University, in 2016. He then worked as a research fellow at Johns Hopkins University. He worked for six years with Texas Instruments-Singapore as a senior Integrated Circuit Design Engineer, designing IPs for mobile processors. His research expertise lies in neuromorphic computing, mixed-signal VLSI systems, computational neuroscience, probabilistic signal processing, and machine learning. His research interest is to understand the signal processing aspects of the brain and apply those to build novel intelligent systems. He is recipients of several awards, such as the Young Investigator Award from Pratiksha Trust, Early Career Research Award by Science and Engineering Research Board, India, Inspire Faculty Award by Department of Science and Technology, India.