URSI AP-RASC, Sydney, Australia, 17 – 22 August 2025

Comparison of Hardware Implementations of Particle Filter and Extrapolated Single Propagation Particle Filter for Tracking of Radio Frequency Interference Sources

Duc Dung VU(1), Sanat K. Biswas(2), Alan Kan(1), and Ediz Cetin(1)
(1) School of Engineering, Macquarie University, Sydney, NSW, Australia
(2) Department of Electronics and Communication Engineering, IIIT Delhi, New Delhi, India

Abstract

Global Navigation Satellite Systems (GNSS) are vulnerable to Radio Frequency Interference (RFI). Prior work has explored the use of Particle Filter (PF) for geo-locating RFI sources, but their high computational cost remains a challenge. This work presents a comparative study of hardware implementations of PF and its efficient variant, Extrapolated Single Propagation Particle Filter (ESP-PF), for RFI source tracking. Results show that the hardware implementation of PF performs slower than ESP-PF and has higher resource consumption. The work highlights the trade-offs between computational complexity and hardware resource utilisation, offering insights for optimised source tracking.

1 Introduction

Many services rely on the timing and positioning data from *Global Navigation Satellite Systems* (GNSS), particularly the *Global Positioning System* (GPS). However, the inherently weak received signal strength of GNSS transmissions makes them susceptible to intentional and unintentional *Radio Frequency Interference* (RFI) [1, 2, 3]. To mitigate this vulnerability, various approaches for detecting, geo-locating, and mitigating RFI sources have been proposed. In the context of geo-localization, the GNSS Environmental Monitoring System (GEMS) [4, 5] employs a hybrid *Angle of Arrival* (AOA) and *Time Difference of Arrival* (TDOA) approach to geo-locate RFI signals with a 1 Hz update rate.

In [6], a conventional *Particle Filter* (PF) was proposed for improved hybrid AOA/TDOA-based RFI geo-localization. However, the high computational cost of PF poses a challenge for field deployment, where power and size are constrained. Various hardware implementations try to address this. In [7], a resampling-optimised PF with a fully parallel architecture was proposed, achieving a 5.62 µs processing time. The sequential nature of computations in the *Sampling* and *Importance Sampling* phases of PF prevents each particle from being computed independently, making this architecture unsuitable for GNSS interference problems. Similarly, other implementations [8, 9, 10] targetting battery power control, radar, and image processing have been proposed, all surpassing the 1 Hz update requirement of GEMS. However, these architectures are all primarily multiple-input multiple-output based and incompatible with the PF in [6], where multivariate computations at all PF phases – *Sampling*, *Importance Sampling*, and *Resampling* – are required, enforcing sequential processing. Thus, a new architecture is needed to optimise PF implementation while meeting the 1 Hz update rate.

In this work, we have selectively parallelised phases of the PF algorithm, aggregating their results for multivariate computation. The hardware implementation, targeting the AMD XCZU28DR *System-on-Chip* (SoC) platform, resulted in $\approx 5\times$ reduction in processing time when compared to the software implementation in [6]. We also compared the PF implementation with the *Extrapolated Single Propagation Particle Filter* (ESP-PF) implementation in [11]. While ESP-PF is an optimised variant of PF, its efficiency gains when implemented in software may not fully translate to hardware due to parallelisation and resource constraints. The comparison results revealed that although ESP-PF outperformed PF in terms of processing speed in hardware, the gain was less significant in software.

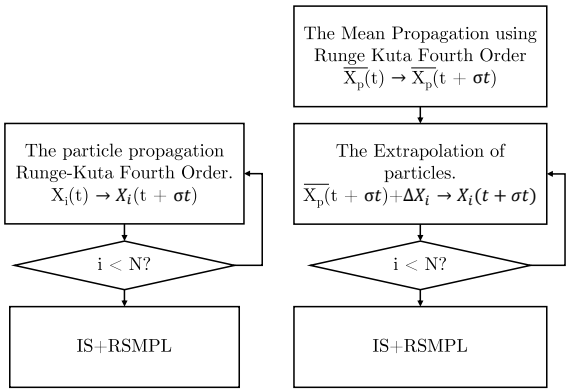
2 Mathematical Model

The PF is used to solve highly nonlinear source tracking problems by approximating a stochastic state vector's probability density function using a finite set of particles, the random sample state vectors [12]. In the RFI source geo-localization context, the PF estimates the position and velocity of the RFI source in addition to the AOA bias, clock bias and drifts between *Sensor Nodes* (SNs). The state vector for this estimation problem is defined as [13]:

$$\mathbf{X}(t) = [\mathbf{r}^T \quad \mathbf{v}^T \quad \mathbf{b_t}^T \quad \mathbf{d_t}^T \quad \mathbf{b_a}^T]^T \tag{1}$$

where $\mathbf{r} = [x \quad y]^T$ and $\mathbf{v} = [\dot{x} \quad \dot{y}]^T$ are the position and velocity vectors of the RFI source, respectively. $\mathbf{b_t} = [b_{t_{12}} \quad b_{t_{13}} \quad b_{t_{23}}]^T$ and $\mathbf{d_t} = [d_{t_{12}} \quad d_{t_{13}} \quad d_{t_{23}}]^T$ are the estimates of clock bias and drift for the i^{th} and j^{th} SNs, respectively, and $\mathbf{b_a} = [b_{a_1} \quad b_{a_2} \quad b_{a_3}]^T$ is the AOA bias at the i^{th} SN. Assuming the clock biases, drifts, and AOA biases are constants, the state space vector model can be written as:

$$\begin{bmatrix} \dot{r} \\ \dot{v} \\ \dot{b}_t \\ \dot{d}_t \\ \dot{b}_a \end{bmatrix} = \begin{bmatrix} A_{4\times4} & 0_{4\times3} & 0_{4\times3} & 0_{4\times3} \\ 0_{3\times4} & 0_{3\times3} & I_{3\times3} & 0_{3\times3} \\ 0_{6\times4} & 0_{6\times3} & 0_{6\times3} & 0_{6\times3} \end{bmatrix} \begin{bmatrix} r \\ v \\ b_t \\ d_t \\ b_a \end{bmatrix} + \begin{bmatrix} 0_{2\times1} \\ a_{2\times1} \\ 0_{9\times1} \end{bmatrix} + v(t) \tag{2}$$



(a) PF Flow Chart (b) ESP-PF Flow Chart

Figure 1. Algorithmic flowchart of (a) PF and (b) ESP-PF. IS + RSMPL are Importance Sampling and Resampling, respectively.

where

$$A_{4\times4} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

and $0_{m\times n}$ is the $m \times n$ zero matrix and $a_{2\times1}$ is the unknown acceleration vector and $v(t)$ is the process noise vector.

The AOA observation θ_i from the i^{th} SN and the TDOA measurement δt_{ij} between the i^{th} and j^{th} SN at the k^{th} epoch can be computed as [1]:

$$\theta(k)_i = tan^{-1}\left(\frac{x(k)-x_i}{y(k)-y_i}\right) + b_{a_i} + \eta_a(k) \tag{4}$$

$$\delta t_{ij} = \tfrac{1}{c}(||\mathbf{r}_i - \mathbf{r}(k)|| - ||\mathbf{r}_j - \mathbf{r}(k)||) + b_{T_{ij}} + kd_{T_{ij}} + \eta_T(k) \tag{5}$$

where η_T and η_a are the TDOA and AOA measurement noise, respectively. The position of the i^{th} or j^{th} SN in x and y coordinates are notated as $x_{i/j}$ and $y_{i/j}$, and the vector $r_{i/j}$ is the position vector of either the i^{th} or j^{th} SN.

The PF flowchart, where N particles are propagated individually at the *Sampling* phase, is shown in Fig. 1(a). The *Importance Sampling* (IS) assigns associated weights to particles, and *Resampling* (RSMPL) re-distributes the particles in the event of degeneracy. The propagation method used in this work is the *Runge–Kutta Fourth Order* (RK4):

$$X(t+\sigma t) = X(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{6}$$

where

$$\begin{aligned} k_1 &= \mathbf{f}(t, X(t)) & k_2 &= \mathbf{f}(t+0.5, X(t)+0.5k_1) \\ k_3 &= \mathbf{f}(t+0.5, X(t)+0.5k_2) & k_4 &= \mathbf{f}(t+0.5, X(t)+k_3) \end{aligned} \tag{7}$$

and $\mathbf{f}$ is the state-space vector model in (2).

As particles are propagated individually, the number of particles dictates the processing time of the PF. To reduce the *Sampling* phase's complexity, the ESP-PF is introduced in [14], where only the *a posteriori* mean state vector is propagated through the RK4, and particles are reconstructed at the next time step using multidimensional Richardson Extrapolation [15]. Since extrapolation is computationally less intensive than RK4 propagation, ESP-PF achieves higher efficiency despite having more steps, as shown in Fig. 1(b). The approximation in ESP-PF is:

$$X_i(t) = \overline{X}_p(t) + e^{\delta t} \times \Delta X_i, \quad i = \{1:N\} \tag{8}$$

where $\overline{X}_p(t)$ is the propagated mean vector, $e^{\delta t}$ is a Jacobian matrix of the state vector in (1). ΔX_i is the random Gaussian Sigmas used for the extrapolation and can be computed with the *a posteriori* mean covariance $\overline{Pxx(t)}$ and r_i is a vector of 13×1 random numbers:

$$\Delta X_i = r_i \sqrt{diag(\overline{Pxx(t)})} \quad i = 1:N \tag{9}$$

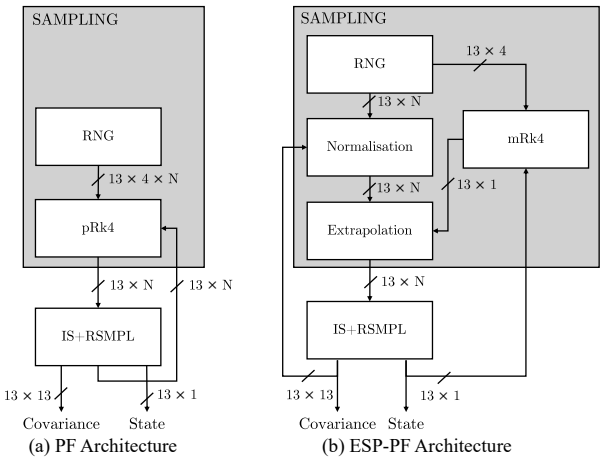
3 Particle Filter Implementation

The PF is implemented on the PL fabric of the XCZU28DR platform, improving the processing time with parallelisation while leveraging the *Processing System* (PS) of the XCZU28DR platform for data transfer and test automation. Further, Vitis HLS and Vitis IDE were used for the hardware implementation. As the PF was originally implemented in MATLAB in [6], it was first converted to C/C++. In [11], hardware implementation of ESP-PF was introduced, which only differs from the PF in the *Sampling* phase. Hence, we leveraged the source code from [11] to implement PF. Hardware architectures of PF and the ESP-PF are shown in Fig. 2, highlighting their similarities (the IS and RSMPL phases) and differences (the *Sampling* phase).

Considering Fig. 2(a), during the *Sampling* phase of PF, particles are propagated individually through the RK4 method (`pRk4`), which requires the execution of the state-space equation, (2), four times as detailed in (6) and (7). Each execution of (2) requires the generation of 13×1 random numbers for estimating the process noise vector **v(t)**, in total 13×4 random numbers are drawn from the `RNG` – *Random Number Generator* – block for one particle propagation. With $1,024$ particles, there will be $13 \times 4 \times 1,024$ random numbers. The `pRk4` blocks use these random numbers to propagate particles from the previous iteration. As there are four executions of the state-space equation, four dedicated hardware blocks are instantiated to enable pipelined execution, optimising particle propagation efficiency. The propagated particles are then assigned weights and re-distributed in the `IS + RSMPL` phase.

For PF implementation, datapaths were quantized to 32-bit fixed-point representation with 11 integer and 21 fractional bits based on the fixed-point analysis result in [11]. The input and output of the PF implementation were stored in the *Block RAMs* (BRAMs) of the PL and transferred to/from the PS through *Advanced eXtensible Interface* (AXI) buses.

The hardware architecture of ESP-PF is shown in Fig. 2(b), where `Normalisation` and `Extrapolation` blocks are



(a) PF Architecture (b) ESP-PF Architecture

Figure 2. Hardware architectures of (a) PF and (b) ESP-PF. Arrows represent data dependency. The grey area shows how the *Sampling* phase for each algorithm is implemented, and N is the number of the particles.

added to compute (8). The `mRk4` block calculates the mean propagation using (6). It is similar to `pRk4` in the PF architecture but only propagates the mean particle. Although the PF architecture is simpler than the ESP-PF one, its computation is more intensive due to the complexity of RK4 propagation, (6), compared to extrapolation, (8). Consequently, iterating across $1,024$ instances of (6) requires more computational effort than that of (8). However, the complex architecture of ESP-PF poses a greater challenge for the Vitis HLS compiler in identifying operation-level pipelining. The compiler cannot determine whether `Normalisation` or `mRk4` outputs arrive first at the `Extrapolation` block, potentially leading to the sequential scheduling of these three blocks. In contrast, `pRk4` in the PF depends only on the `RNG` block, inherently forming a pipeline structure that the HLS compiler can efficiently schedule. The PF architecture also demands extra hardware resources to store input particles and $13 \times 4 \times N$ generated random numbers for `pRk4`, where N is the number of particles. Moreover, its propagation is computationally more complex than ESP-PF extrapolation, making its parallelisation more resource-intensive and less scalable.

4 Results and Discussion

Two hundred observation data points from the same field trial that was used in [11] were utilised to evaluate the position estimation accuracy of the PF and the ESP-PF with $1,024$ particles as software (SW) running in MATLAB and hardware (HW) implemented on the XCZU28DR platform, using 32-bit fixed-point representation. Position estimation performance was assessed over 100 Monte Carlo runs by analysing the error between the estimated and actual source locations. Hardware performance in terms of resource utilisation and execution speed was recorded to assess each algorithm's suitability for hardware implementation. The PF and ESP-PF hardware implementations were run on

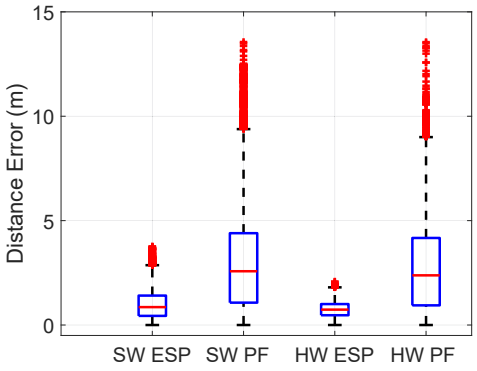


Figure 3. Position estimation errors for PF and ESP-PF implemented in software (SW) and Hardware (HW). The central mark in each box represents the median, and the box edges are the 25^{th} & 75^{th} percentile. The whiskers show the range of the data, and red $+$ symbols represent the outliers, which are $\approx 2.7\sigma$ away from the median.

the XCZU28DR platform with a clock speed of 200 MHz, while the software implementations were executed on an Intel Core i7-13700H processor using MATLAB R2023b.

Position estimation performance of the PF and the ESP-PF SW and HW implementations is shown in Fig. 3. In agreement with what was reported in [11], the hardware implementation of ESP-PF, HW ESP, demonstrates better estimation performance, particularly the convergence, compared to its MATLAB counterpart, SW ESP, due to quantization effects. The SW ESP has an *Inter-Quartile Range* (IQR) of 0.44 – 1.41 m with 194 outliers, whereas the HW ESP achieves a reduced IQR of 0.47 – 1.03 m with 48 outliers. However, such improvement is not observed in the PF. The IQR and the number of outliers in HW PF are higher than in SW PF, with an IQR of 1.05 – 5.11 m and $1,577$ outliers versus 1.09 – 4.79 m and 738 outliers, respectively.

Table. 1 compares the resource utilisation and processing time of the PF and ESP-PF targetting the XCZU28DR platform. As can be observed, in general, the resource utilisation of the PF is significantly higher than that of the ESP-PF, especially in terms of the number of DSP slices required, increasing by 19.3%. Other resources, BRAM, *Look Up Tables* (LUTs), and *Flip Flops* (FFs) are 10.8%, 6.7%, and 16.2% higher than those of the ESP-PF, respectively.

	HW ESP-PF [11]	HW PF
LUTs(k)	125.0 (31.07%)	134.0 (33.25%)
FFs(k)	143.0 (16.91%)	153.3 (18.06%)
DSPs	600 (14.04%)	716 (16.76%)
BRAMs	290 (26.85%)	325 (30.09%)

Table 1. Hardware resource utilisation of PF and ESP-PF on the XCZU28DR platform.

Table 2 depicts the processing times for the SW and HW implementations of the PF and ESP-PF. While SW PF requires up to 512 ms for a single estimation, SW ESP-PF

Works	SW ESP [14]	SW PF [6]	HW ESP [11]	HW PF
Speed (ms)	123	512	77.36	107.68

Table 2. Processing time of the PF and ESP-PF.

only takes 123 ms. This speed advantage stems from the ESP-PF's optimisation using multidimensional Richardson extrapolation. However, this strong advantage did not carry over to the hardware implementation, where HW PF takes 107.68 ms compared to 77.35 ms for HW ESP. This difference may come from the complex architecture of the ESP-PF, which could hinder the Vitis HLS compiler's ability to optimise the synthesized design.

5 Conclusion

Hardware implementation of a PF for RFI source geolocalization is presented, and its performance is evaluated in terms of position estimation, hardware utilisation and processing speed and compared with that of the ESP-PF. The hardware implementation of the PF achieved a processing time of 107.68 ms, which is 9× faster than the 1 Hz update rate requirement of GEMS. Further, it achieved a position estimation error IRQ of 1.09 – 4.79 m.

It was observed that, unlike ESP-PF, the hardware implementation of PF did not benefit from word-length quantisation. Additionally, while the ESP-PF offered a considerable performance gain relative to PF when implemented in software, this was not reflected in the hardware implementation results. Identifying the factors behind these results will guide future optimisation efforts.

References

[1] A. G. Dempster and E. Cetin, "Interference localization for satellite navigation systems," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1318–1326, 2016.

[2] D. Borio, F. Dovis, H. Kuusniemi, and L. L. Presti, "Impact and detection of gnss jammers on consumer grade satellite navigation receivers," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1233–1245, 2016.

[3] L.-T. Hsu, Y. Gu, Y. Huang, and S. Kamijo, "Urban pedestrian navigation using smartphone-based dead reckoning and 3-d map-aided gnss," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1281–1293, 2015.

[4] E. Cetin, R. Thompson, M. Trinkle, and A. Dempster, "Interference detection and localization within the GNSS environmental monitoring system (GEMS) — System update and latest field test results," in *Proc. 27th International Technical Meeting of Satellite Division of Institute of Navigation (ION GNSS+ 2014)*, Sept. 2014, pp. 3449–3460.

[5] E. Cetin, R. J. Thompson, and A. G. Dempster, "Passive interference localization within the GNSS environmental monitoring system (GEMS): TDOA aspects," *GPS solutions*, vol. 18, no. 4, pp. 483–495, 2014.

[6] S. K. Biswas and E. Cetin, "Particle Filter based Approach for GNSS Interference Source Tracking: A Feasibility Study," in *2020 XXXIIIrd General Assembly and Scientific Symposium of the International Union of Radio Science*, 2020, pp. 1–4.

[7] A. Krishna, A. Van Schaik, and C. S. Thakur, "FPGA implementation of particle filters for robotic source localization," *IEEE Access*, vol. 9, pp. 98 185–98 203, 2021.

[8] Y. Song, D. Liu, and Y. Peng, "FPGA-based implementation of lithium-ion battery soh estimator using particle filter," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.

[9] 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.

[10] T. Chisholm, R. Lins, and S. Givigi, "FPGA-based design for real-time crack detection based on particle filter," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5703–5711, 2019.

[11] D. D. Vu, S. K. Biswas, A. Kan, and E. Cetin, "Hardware implementation of a particle filter for GNSS interference source localization," in *2024 22nd IEEE Interregional NEWCAS Conference (NEWCAS)*, 2024, pp. 70–74.

[12] M. 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, August 2002.

[13] S. K. Biswas and E. Cetin, "GNSS Interference Source Tracking using Kalman Filters," in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 2020, pp. 877–882.

[14] S. K. Biswas, L. Qiao, and A. G. Dempster, "A novel a priori state computation strategy for the unscented kalman filter to improve computational efficiency," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1852–1864, 2017.

[15] S. K. Biswas and A. G. Dempster, "Approximating sample state vectors using the espt for computationally efficient particle filtering," *IEEE Transactions on Signal Processing*, vol. 67, no. 7, pp. 1918–1928, 2019.