International Journal of Engineering

Journal Homepage: www.ije.ir

Improved Distributed Particle Filter Architecture with Novel Resampling Algorithm for Signal Tracking

Z. Talebi[a], S. Timarchi*[b]

[a] *Faculty of Electrical Engineering, Shahid Beheshti University, Tehran, Iran*
[b] *Faculty of Electrical Engineering, Shahid Beheshti University, Tehran, Iran; and School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*

PAPER INFO

Paper history:
Received 27 July 2019
Received in revised form 01 September 2020
Accepted 03 September 2020

Keywords:
Particle Filter
Independent Metropolis
Hasting Resampling
FPGA
Signal Tracking

ABSTRACT

Resampling is a critical step in Particle Filter (PF) because of particle degeneracy and impoverishment problems. Independent Metropolis Hasting (IMH) resampling algorithm is a robust and high-speed method that can be used as the resampling step in PF. In this paper, a new algorithm based on IMH resampling is first proposed. The proposed algorithm classifies the particles before entering to the resampling module. The classification causes those essential particles are only routed to the IMH resampler. Then we propose a distributed architecture to reduce the execution time and high-speed processing for resampling. Simulation results for tracking a signal indicate that the PF with the proposed resampling architecture has acceptable tracking performance in comparison to other resampling methods. The PF architecture with the novel Improved IMH (IIMH) resampling algorithm has 33% more speed than the best-reported method in PF. Also, the proposed distributed PF architecture achieve 79% more speed compared with the best-reported method in PF. FPGA-based implementation results indicate that the utilization of the proposed IIMH resampling algorithm in PF and also distributed architecture lead to hardware resource and area usage reduction.

doi*: 10.5829/ije.2020.33.12c.07*

1. INTRODUCTION

Object tracking through multiple cameras is a popular research topic in security and surveillance systems especially when human objects are the target [1]. Utilizing adaptive filters is a dominant solution for visual tracking problems [2]. The signal tracking in Particle Filter (PF) is performed by searching the space of the states with randomly generated samples called particles. PF consists of three steps, 1) particle sampling, 2) weight calculation, and 3) resampling. Without resampling, PF suffered from a degeneracy problem, which means that a few particles with higher weights participate in the resampling and those particles with lower weights will be discarded [3-7]. Resampling, however, may introduce undesired effects. One of them is sample impoverishment.

Two main types of resampling methods used in PF are sequential resampling and compound resampling. Sequential resamplings are standard methods that have high complex computation load. Systematic Resampling (SR) [8-10], Residual Resampling (RR) [11-14], Residual-Systematic Resampling (RSR) [3] are the three most common examples for sequential resampling. Compound resampling operates based on grouping the particles. The grouping can be done by one or more threshold values or based on the ratio between particle weights. The thresholds can be fixed, variable, or adaptive [7].

Independent Metropolis Hasting (IMH) algorithm in one of the robust and high-speed compound resampling methods that works based on the ratio between the particle weights. This method works as soon as the first particle and its corresponding weights are produced and it doesn’t wait until whole particles to be generated.

*Corresponding Author Institutional Email: *s_timarchi@sbu.ac.ir* (S. Timarchi)

By using multiple Processing Elements (*PEs*) and a Central Unit (*CU*) we can process the particles simultaneously. This is known as Distributed resampling methods. The methods include Centralized resampling, resampling with proportional allocation, resampling with non-proportional allocation and they use two types of resampling, one resampling inside each *PE* called *intra-resampling* and another between two *PEs* called *inter-resampling*.

In this paper, a new IMH algorithm and architecture, called Improved IMH (IIMH) to increase the processing speed in IMH resampler is proposed that is derived from both threshold-based resampling methods and the ratio between the weights. Moreover, Distributed IIMH (DIIMH) architecture with 4 *PEs* to process the particles in parallel is explored in this paper. Results of tracking a random signal show that the proposed IIMH algorithm can have the same accuracy of standard SR. Comparison results of hardware resource utilization and processing speed show that proposed architectures have acceptable performance in comparison to existing efficient methods reported in the literature.

The rest of the paper is organized as follows: In Section 2, a literature review and a brief theory of PF and resampling are presented, and then, we review the IMH and Modified IMH (MIMH) resamplers. In Section 3, the proposed IIMH algorithm and architectures, as well as DIIMH architecture are presented. In Section 4, the comparison and results of the proposed methods are discussed. Section 5 concludes the paper.

2. LITERATURE REVIEW ON PF AND RESAMPLING METHODS

PFs are used for tracking states of dynamic state-space models described by the following equation [8]:

$$x_n = f(x_{n-1}) + u_n; \qquad y_n = g(x_n) + v_n \tag{1}$$

where x_n is the state vector of target position, y_n is a vector of observations, u_n and v_n are independent noise vectors with known distributions. PFs accomplish tracking of x_n by updating a random measure $\{x_{1:n}^{(m)}, w_n^{(m)}\}_{m=1}^{M}$ which is composed of M particles $x_n^{(m)}$ and their weights $w_n^{(m)}$ defined at time instant n, recursively [3]. The random measure a page no. proximates the posterior density of the unknown trajectory $x_{1:n}$, $p(x_{1:n}|y_{1:n})$, where $y_{1:n}$ is the set of observations. After resampling, the next particles are more concentrated in domains of the higher posterior probability. The PF operations are performed according to the following steps:

1. Generation of particles (samples) $x_n^{(m)} \sim \pi(x_n|x_{n-1}^{(t_{(n-1)}^{m})}, y_{1:n})$, where $\pi(x_n|x_{n-1}^{(t_{(n-1)}^{m})}, y_{1:n})$ is an importance density and $i_n^{(m)}$ is an array of indexes, which shows that the particle m should be reallocated to the position $i_n^{(m)}$;
2. computation of weights by:

$$w_n^{*(m)} = \frac{w_{n-1}^{(t_{(n-1)}^{m})}}{a_{n-1}^{(t_{(n-1)}^{m})}} \frac{p(y_n|x_n^{(m)})p(x_n^{(m)}|x_{n-1}^{(t_{(n-1)}^{m})})}{\pi(x_n^{(m)}|x_{n-1}^{(t_{(n-1)}^{m})}, y_{1:n})} \tag{2}$$

3. Resampling $i_n^{(m)} \sim a_n^{(m)}$, where $a_n^{(m)}$ is a suitable resampling function for the particle $x_n^{(m)}$.

Different classifications of resampling methods exist [3, 8, 11]. Recently, Li et al. [7] propose a complete classification for resampling methods. For reducing hardware complexity, the *RR* algorithm offers interesting features for fixed-point implementation presented in literature [5]. The CR architecture consists of two loops. The first loop selects the substantial particles, and the second one multiplies the selected particles sequentially [12].

Recently, most of the researches are tended to distributed architecture for PF. The main reason is to minimize the execution time through paralleling and pipelining of operations.

Most of the resampling algorithms can start only after all particles are generated and then cumulative sum and normalized weights start to be calculated. This fact is a bottleneck in the pipelined implementation. To remove this weakness, some resampling algorithms operate based on the ratio between the weights, like Metropolis Hastings (MH) [13] and IMH algorithm [14]. These methods do not need to normalize weights and therefore they are suitable for parallel processing. The details of the IMH algorithm is described in literature [14]. A Modified IMH (MIMH) algorithm is proposed in literature [15]. In the IMH algorithm, the new particle will be retained when $u < \alpha(\hat{x}_t^{j-1}, x_t^{j})$, which can be simplified as follows:

$$u \le \alpha(\hat{x}_t^{(j-1)}, x_t^{(j)}) = min\{\frac{w(x_t^{(j)})}{\hat{x}_t^{(j-1)}}, 1 \le \frac{w(x_t^{(j)})}{\hat{x}_t^{(j-1)}} \tag{3}$$

and this equation in the MIMH algorithm is equal as follows:

$$u \times w(\hat{x}_t^{(j-1)}) \le w(x_t{}^{(j)}) \tag{4}$$

If the weight of the new particle is larger than the product of the uniform random-number u and the weight of the last accepted particle in the chain, the new particle is selected, otherwise, the last accepted particle is repeated once more.

3. PROPOSED IIMH ALGORITHM

3. 1. Proposed IIMH Resampling Algorithm

The proposed IIMH algorithm can be defined as Algorithm 1. In the first step, two threshold values T1 and

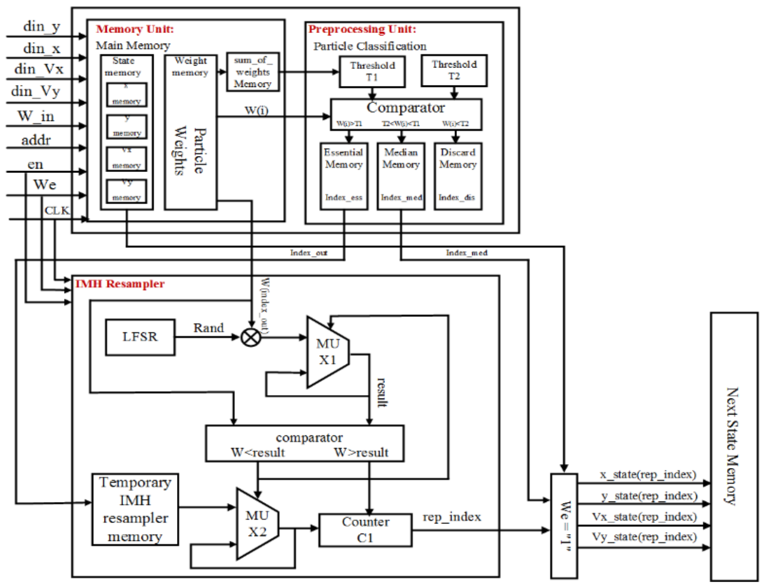

Figure 1. Proposed architecture for IIMH resampling in PF

T2 are calculated. These values are based on particle weights. Then the weight of the particles is compared with the two threshold values in the second step. According to comparison results, the particles are grouped into three different categories as below:

1. Essential particles (N_{ess}): Particles whose weights are greater than $T1$, are assumed as essential particles. These high weight particles are used as inputs of the resampler module.
2. Median particles (N_m): The particles whose weights are less than $T1$ and greater than $T2$ are assumed in this group. These particles are not replicated, but they apage no. ear in outputs of the resampler module. These particles included essential information about the target trajectory.
3. Discarded particles (N_d): These particles whose weights are less than $T2$ are considered as discarded particles. It means that they are low weight particles and don't have any useful information about the trajectory.

After classifying the particles, the indexes of essential particles are used for resampling. In the third step, the first essential particle will be used as an initialized value in the chain. In step 4, a Linear Feedback Shift Register (LFSR) will be used to generate a uniform random number u as a resampling function [16].

Algorithm 1. Proposed IIMH algorithm

```
$\{\tilde{x}_k^{(j)}\}_{j=1}^{N}$ =IMH-Sampler $\{x_k^{(j)}, w_k^{(j)}\}_{j=1}^{N-N_m-N_d+N_b}$
1. Threshold T1 and T2 calculation:
        $T1 = \frac{\sum_{i=1}^{M} w_i}{M}, \quad T2 = \frac{T1}{2}$
2. Classify the particles into 3 groups:
$N_{ess}: T1 \leq w(i) \quad for \quad i = 1,2,\dots,N$
$N_m: T2 \leq w(i) \leq T1 \quad for \quad i = 1,2,\dots,N$
$N_d: w(i) \leq T2 \quad for \quad i = 1,2,\dots,N$
3. Initialize the chain with $\hat{x}_k^{(1)} = x_k^{(1)}$. The first particle is accepted as seed.
4. For $j = 2,3,\dots,N - N_m - N_d + N_b$
    - Calculate the acceptance probability:
    $\alpha(\hat{x}_k^{(j-1)}, x_k^{(j)}) = \min\{\frac{w(x_k^{(j)})}{w(\hat{x}_k^{(j-1)})}, 1\}$ where $x_k^{(j)}$ represents the new particles, $\hat{x}_k^{(j-1)}$ is the last accepted particle and $w(x_k^{(j)})$ is the associated weight of the particle $x_k^{(j)}$.
    - Generate a uniform random-number as a resampling function:
    $u \sim u[0,1]$
    - Determining the accepted and removed particles:
    $\hat{x}_k^{(j)} = \begin{cases} x_k^{(j)}, & if \quad u \times w(\hat{x}_t^{(j-1)}) \leq w(x_t^{(j)}) \\ \hat{x}_k^{(j-1)}, & otherwise\,. \end{cases}$
5. Discard the first $N_b$ samples for burn-in particles and keep $N - N_m - N_d$ samples:
$\{\tilde{x}_k^{(j)}\}_{j=1}^{N-N_m-N_d} = \{\hat{x}_k^{(j)}\}_{j=N_b+1}^{N-N_m-N_d+N_b}$
```

3. 2. Proposed IIMH Resampling Architecture

Proposed IIMH resampling architecture is shown in Figure 1.

The proposed architecture contains a memory unit and a preprocessing unit. Before resampling, the four-element vectors (x, y, V_x, V_y) are loaded into state memory separately and their associated weights are loaded into the weight memory as parts of the memory unit. The weight memory consists of a register file with a size of $M \times w$ for storing the weights of each particle, where w represents the length of weight value.

The preprocessing unit contains the particle classification module, where the two threshold values are calculated as described in the first step of Algorithm 1. As the weights of particles are loaded into weight memory, the cumulative sum of weights is calculated in *sum_of_weights* memory, simultaneously with calculating T1 and T2. As T1 and T2 are calculated, the weight of the particles is compared with these two threshold values according to the results of the comparison, the particles are loaded in three different memory are named as *Essential memory*, *Median memory*, and *Discard memory*. After classifying the particles, the index of essential particles is routed into the IMH resampler module. Instead of storing four-element vectors of particles in this scheme, their indexes are only stored in the memory and routed in this scheme, because they have a similar addressing index. So memory size $M \times 4$ is substituted by memory with size $M \times 1$.

3. 3. Proposed Distributed Architecture For IIMH

By employing distributed architecture and several *PEs*, the particles are divided and assigned to *PEs* to be processed in parallel. The execution time in distributed architecture for IIMH (DIIMH) would be reduced to the number of *PEs*. By choosing a large number of *PEs*, hardware resource utilization would be increased. So to achieve a trade-off between hardware resource usage and execution time, choosing a page no. proximates number of *PEs* is essential. Figure 2 shows DIIMH architecture with 4 *PEs* and 1 *CU*.

Each *PE* in our architecture is similar to the IMH resampler shown in Figure 1. The function of *CU* is to collect the partial sums of the weights from *PEs* to calculate the output weights and finally perform the *inter-resampling*. If the number of particles produced by each *PE* produces is not equal to the number of the particles that *CU* reports, median particles would be routed to the output of *CU* from $(N_{ess}+1)^{th}$ address of *next_state_memory*.

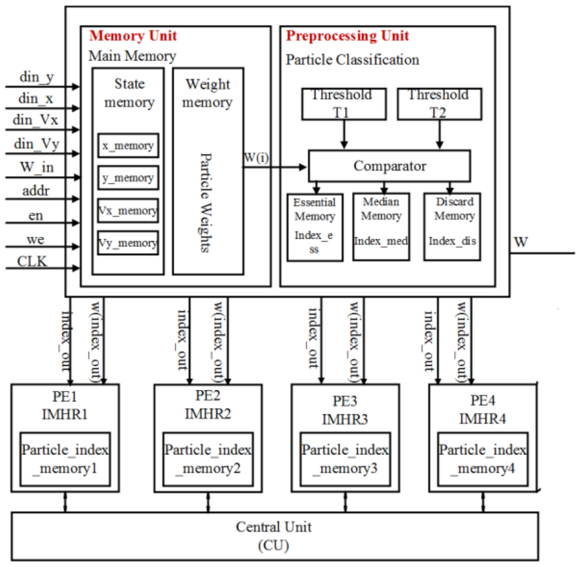

Figure 2. Proposed DIIMH resampler architecture

4. RESULTS AND COMPARISON

4. 1. Signal Tracking Performance

The tracking performance of the proposed IIMH resampling algorithm in PF has been studied for random signals. Figure 3 and Figure 4 show the tracking result of the PF with the proposed IIMH algorithm for $\boldsymbol{M = 32}$ and $\boldsymbol{M = 1024}$ particles, respectively.

As we know, tracking a random signal with PF depends on several parameters. One of the most important factors is the number of particles. It can be observed that in the above figures, increasing the number of particles leads to more accuracy in random signal tracking. As the number of particles entering to the PF with the proposed IIMH algorithm increases, the amount of tracking error decreases.

The tracking result of the PF with the proposed IIMH algorithm in comparison to PF with the Systematic Resampling (SR) algorithm is shown in Figure 5.

To determine the quality of signal tracking depicted in Figure 5, Root Mean Squared Error (RMSE) is measured for two algorithms. *RMSE* is a mixture measure that reflects the bias and variance of PF estimation [4]. In our design, the output state vector consists of coordinate components *x* and *y* and the weight component *w*. From Figure 6 it is obvious that the *RMSE* value of the proposed architecture is lower than the SR algorithm.

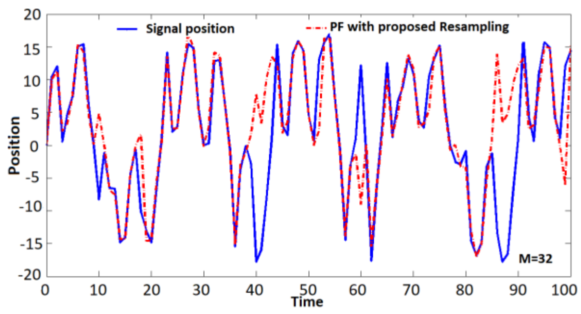

Figure 3. Tracking a random signal with a proposed IIMH algorithm in PF with M=32 particles

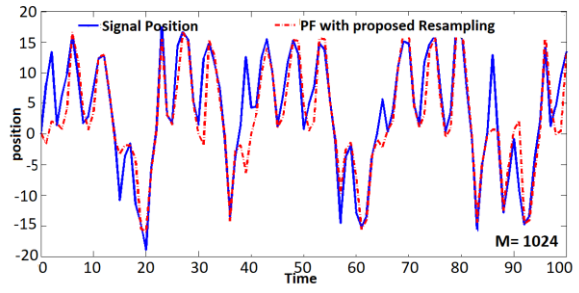

Figure 4. Tracking a random signal with a proposed IIMH algorithm in PF with M=1024 particles

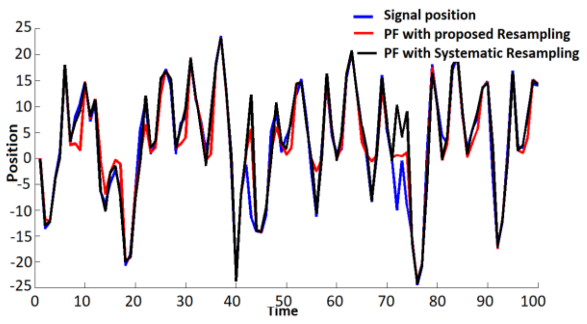

Figure 5. Tracking a random signal with a proposed IIMH algorithm and SR algorithm in PF with M=1024 particles

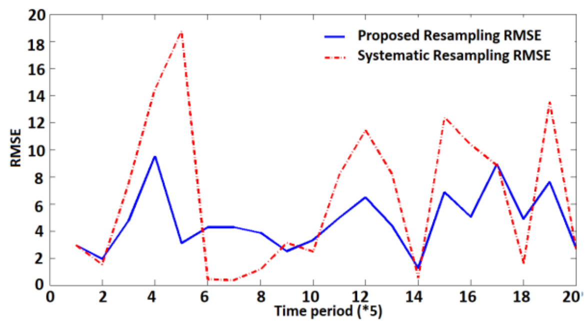

Figure 6. RMSE values of the proposed IIMH algorithm and SR algorithm in PF

Tha average *RMSE* value of the proposed IIMH method is about 25% lower than the SR algorithm that demonstrates the high quality of the proposed method.

4. 2. Execution Time Figure 7 shows the execution timing diagram of PF with the proposed IIMH resampling architecture. The overall execution time for PF can be achieved as below:

$$T_{proposed} = (L_c + L_w + L_s + L_{IMH} + M - N_d - N_m)\, T_{clk} \quad (5)$$

where L_c, L_w, L_s, L_{IMH} represent the startup latencies of the particle classification, weight calculation, sampling, IMH resampler, respectively. N_d and N_m are the numbers of discarded particles and median particles, respectively. T_{clk} is the system clock period. M is the total number of particles.

The overall execution time for PF with proposed DIIMH architecture can be achieved as below:

$$T_{DIIMH} = (L_c + K + L_I + L_{IMH} + (\frac{M-N_d-N_m}{K}) + L_w + L_s)\, T_{clk} \quad (6)$$

Tables 1 and 2 show the resampling frequency comparison for proposed IIMH and DIIMH resampling architectures and other hardware-based resampling methods at a clock frequency of 60 MHz. From the tables, it is concluded that the proposed IIMH and DIIMH methods suggest an acceptable speed in comparison to other resampling methods in PF. The comparison result shows that the proposed IIMH and DIIMH architecture have 33% and 79% more speed than the best reported methods in single *PE* and 4 *PEs*, respectively.

4. 3. Resource Utilization Table 3 shows the resource utilization of proposed IIMH and DIIMH and the existing resampling methods on a Xilinx Virtex5 FPGA (XC5VSX50T) platform as an example.

It is observed that the proposed IIMH and DIIMH reduce the number of DSP units, Block RAM, LUTs, slice FFs and Registers in comparison to other methods. Distributed resampling architectures with more *PEs* consume more device utilization. However, the performance of distributed architectures is better than

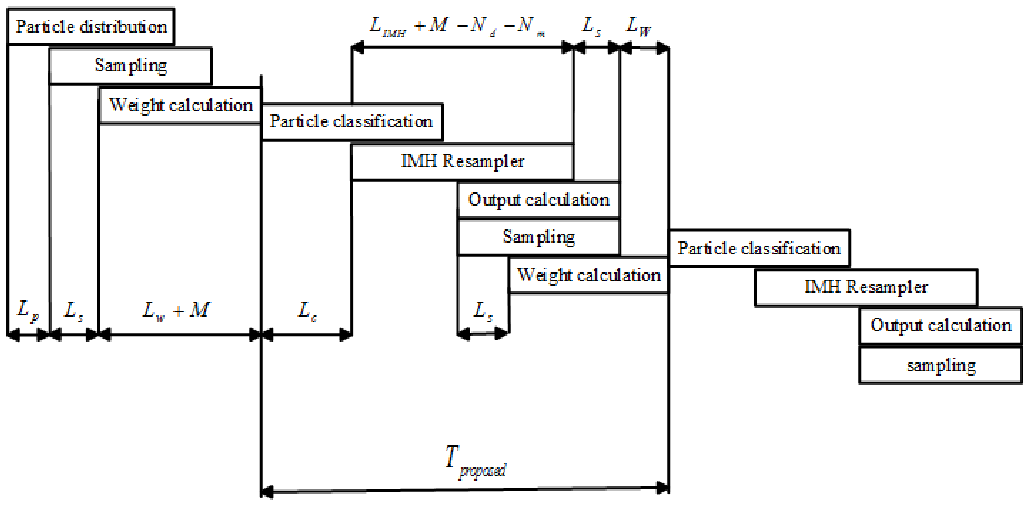

Figure 7. Execution time of PF with proposed IIMH architecture

TABLE 1. Resampling frequency of PF at $f_{clk} = 60$MHz with single *PE* for different resampling methods

Resampling method	Resampling frequency
Proposed IIMH	60 KHz
HR [9]	27 KHz
RSR [6]	45 KHz
RR [5]	19.5 KHz
SR [12]	9.6 KHz
CR [12]	14.4 KHz

TABLE 2. Resampling frequency of PF at $f_{clk} = 60$MHz with 4 *PEs* for distributed resampling methods

Resampling Method	Resampling Frequency
Proposed DIIMH	174 KHz
Distributed IMH [15]	50 KHz
Distributed HR [9]	97 KHz
Distributed RSR [6]	64 KHz
Distributed RR [12]	40 KHz

TABLE 3. Resource utilization of resampling methods implemented on Xilinx Virtex5 (XC5VSX50T)

Resampling methods	Slice register	Slice LUTs	Slice FFs	Block RAM	DSP48Es
Proposed IIMH	2728	4398	3662	12	7
Proposed DIIMH	10712	18395	15641	17	27
IMH [15]	19350	28065	26474	48	101
Distributed HR [9]	11883	20607	16320	28	30

resampling methods with single *PE*. Resource usage reduction of the proposed architectures is due to two main reasons; The first reason is because of using the index of particles instead of using four-element input vectors. The second reason is that a simple routing of *CU* is employed in the proposed DIIMH scheme. Therefore this scheme doesn’t need large temporary memories with high capacity in *PEs* and *CU*.

5. CONCLUSION

In this paper, an improved algorithm and an efficient architecture for IMH resampling, namely IIMH were first proposed. The algorithm is based on classifying the particles before assigning them to the resampling module. The technique would speed up the resampling step by considering only the essential particles in PF. Afterward we proposed a distributed architecture for IIMH, namely DIIMH to increase the processing speed through parallel processing. The results show that resampling frequencies of IIMH and DIIMH methods are 60 KHz and 174 KHz that they are about 33% and 79% more than the best reported methods. Also, we can achieve about 39% reduction in RAM usage due to the simple designing of *CU* in comparison to the most efficient hardware-based method.

6. ACKNOWLEDGMENT

The authors would like to thank the Institute for Research in Fundamental Sciences (IPM) which supported this research in the context of research project number CS1396-4-670.

7. REFERENCES

1. Feizi, A., "Convolutional gating network for object tracking", ***International Journal of Engineering***, ***Transactions A: Basics,*** Vol. 32, No. 7, (2019), 931-939. DOI: 0.5829/IJE.2019.32.07A.05
2. Sadegh Moghadasi, S. and Faraji, N., "An efficient target tracking algorithm based on particle filter and genetic algorithm", ***International Journal of Engineering***, ***Transactions A: Basics,*** Vol. 32, No. 7, (2019), 915-923. DOI: 10.5829/IJE.2019.32.07A.03
3. Liu, J.S., Chen, R. and Logvinenko, T., A theoretical framework for sequential importance sampling with resampling, in Sequential monte carlo methods in practice. 2001, Springer.225-246. DOI: 10.1007/978-1-4757-3437-9_11
4. Zhao, Z., Wang, T., Liu, F., Choe, G., Yuan, C. and Cui, Z., "Remarkable local resampling based on particle filter for visual tracking", ***Multimedia Tools and Applications***, Vol. 76, No. 1, (2017), 835-860. DOI: 10.1007/s11042-015-3075-6
5. Hong, S., Chin, S.-S., Djurić, P.M. and Bolić, M., "Design and implementation of flexible resampling mechanism for high-speed parallel particle filters", ***Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology***, Vol. 44, No. 1-2, (2006), 47-62. DOI: : 10.1007/s11265-006-5919-9
6. Abd El-Halym, H.A., Mahmoud, I.I. and Habib, S., "Proposed hardware architectures of particle filter for object tracking", ***EURASIP Journal on Advances in Signal Processing***, Vol. 2012, No. 1, (2012), 17. DOI: 10.1186/1687-6180-2012-17
7. Li, T., Bolic, M. and Djuric, P.M., "Resampling methods for particle filtering: Classification, implementation, and strategies", ***IEEE Signal Processing Magazine***, Vol. 32, No. 3, (2015), 70-86. DOI: 10.1109/MSP.2014.2330626
8. Bolić, M., Djurić, P.M. and Hong, S., "Resampling algorithms for particle filters: A computational complexity perspective", ***EURASIP Journal on Advances in Signal Processing***, Vol. 2004, No. 15, (2004), 403686. DOI: 10.1155/S1110865704405149
9. Pan, Y., Zheng, N., Tian, Q., Yan, X. and Huan, R., "Hierarchical resampling algorithm and architecture for distributed particle filters", ***Journal of Signal Processing Systems***, Vol. 71, No. 3, (2013), 237-246. DOI: 10.1007/s11265-012-0712-4
10. Gan, Q., Langlois, J.P. and Savaria, Y., "A parallel systematic resampling algorithm for high-speed particle filters in embedded

systems", ***Circuits, Systems, and Signal Processing***, Vol. 33, No. 11, (2014), 3591-3602. DOI: 10.1007/s00034-014-9820-7

11. Douc, R. and Cappé, O., "Comparison of resampling schemes for particle filtering", in ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, IEEE. (2005), 64-69. DOI: 10.1109/ISPA.2005.195385
12. Hong, S.-H., Shi, Z.-G., Chen, J.-M. and Chen, K.-S., "A low-power memory-efficient resampling architecture for particle filters", ***Circuits, Systems and Signal Processing***, Vol. 29, No. 1, (2010), 155-167. DOI: 10.1007/s00034-009-9117-4
13. Murray, L., "Gpu acceleration of the particle filter: The metropolis resampler", arXiv Preprint arXiv:1202.6163, (2012).
14. Sankaranarayanan, A.C., Srivastava, A. and Chellappa, R., "Algorithmic and architectural optimizations for computationally efficient particle filtering", ***IEEE Transactions on Image Processing***, Vol. 17, No. 5, (2008), 737-748. DOI: 10.1109/TIP.2008.920760
15. Hong, S., Shi, Z. and Chen, K., "Easy-hardware-implementation mmpf for maneuvering target tracking: Algorithm and architecture", ***Journal of Signal Processing Systems***, Vol. 61, No. 3, (2010), 259-269. DOI: 10.1007/s11265-010-0450-4
16. Medina, A.R., "Hardware-based particle filter with evolutionary resampling stage", Master thesis, 3-2014, Universidad Politécnica de Madrid, (2014).

Persian Abstract

چکیده

نمونه‌برداری مجدد به دلیل انحطاط و ضعف ذرات یک مرحله ضروری در فیلتر ذره است. الگوریتم نمونه برداری مجدد Independent Metropolis Hasting (IMH) یک روش قوی و پر سرعت است که می‌تواند در مرحله نمونه‌برداری مجدد در فیلتر ذره مورد استفاده قرار گیرد. در این مقاله ابتدا یک الگوریتم جدید بر اساس نمونه‌برداری مجدد IMH پیشنهاد شده است. الگوریتم پیشنهادی ذرات را قبل از ورود به ماژول نمونه‌برداری مجدد طبقه‌بندی می‌کند. این طبقه‌بندی باعث می‌شود که تنها ذرات ضروری به واحد نمونه‌بردار مجدد IMH وارد شود. سپس معماری توزیع شده ای به منظور کاهش زمان اجرا و پردازش سرعت بالا برای نمونه‌برداری مجدد ارائه شده است. نتایج شبیه‌سازی برای ردیابی یک سیگنال نشان می‌دهد که فیلتر ذره با معماری نمونه‌برداری مجدد پیشنهادی دارای عملکرد ردیابی قابل قبول در مقایسه با سایر روش‌های نمونه‌برداری مجدد است. معماری فیلتر ذره با الگوریتم نمونه‌برداری مجدد پیشنهادی IIMH دارای ۳۳ ٪ سرعت بیشتر در مقایسه با بهترین روش گزارش داده شده در فیلتر دره است. همچنین معماری فیلتر ذره توزیع شده‌ی پیشنهادی دارای ۷۹ ٪ سرعت بیشتر در مقایسه با بهترین روش گزارش داده شده است. نتایج پیاده‌سازی بر اساس FPGA نشان می‌دهد که استفاده از الگوریتم نمونه‌برداری مجدد پیشنهادی در فیلتر ذره و همچینین معماری توزیع شده منجر به کاهش منابع سخت افزاری و مساحت مورد استفاده می شود.