PARTICLE STATE COMPRESSION SCHEME FOR CENTRALIZED MEMORY-EFFICIENT PARTICLE FILTERS

Qinglin Tian[1], Yun Pan[2], Xiaolang Yan[1], Ning Zheng[3], Ruohong Huan[4]

[1]College of Electrical Engineering, Zhejiang University, China
[2]Department of Information Science & Electronic Engineering, Zhejiang University, China
[3]Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, USA
[4]College of Computer Science and Technology, Zhejiang University of Technology, China

ABSTRACT

In this paper, particle state compression scheme is proposed together with its architecture for centralized implementation of particle filters. In the scheme, state values are processed in original bit-width, stored in a compressed way and recovered before sampling of next iteration. The advantage of the scheme is that particle states memory requirement can be greatly reduced while the trade-off is the deviations between original and recovered states introduced by the process. A case study in Nearly Constant Turn (NCT) scenario shows that while achieving the same level of filtering accuracy, proposed scheme can save up to 49.69% memory overhead for storing particle state values compared to traditional realizations.

Index Terms—particle filters, compression scheme, memory-efficient, implementation.

1. INTRODUCTION

In nonlinear/non-Gaussian Bayesian state estimation problems, particle filter (PF) [1] based on sequential Monte Carlo method outperforms traditional extended Kalman filter (EKF) and has gained popularity in fields including signal processing, computer vision and navigation [2, 4]. The key idea is to recursively approximate the probability density function (PDF) using a set of random samples by sampling, weight calculation and resampling steps. Due to its high computational complexity nature, dedicated hardware is a more promising solution other than software when implementing PFs. [3] studies the effect of finite precision processing in hardware implementation and shows that long bit-width is needed to achieve satisfying filtering performance. The resulting hardware complexity and considerable memory overhead all present challenges for a more efficient design. Previous works made great efforts on algorithmic [5-7] and architectural [5, 8, 9] level of particle filtering while the astonishing memory overhead for particle states keep unchanged. In this paper, particle state compression scheme and its architecture are proposed with the purpose of reducing memory requirement for particle states while maintaining filtering accuracy, which is not considered in earlier studies. In the scheme, particles are processed in original bit-width, stored in a compressed way and recovered before sampling of next iteration. Memory requirement is reduced with the trade-off of deviations between original and recovered state values. A case study by simulation is demonstrated to help analyzing the influence of the deviations as well as proving the effectiveness of the proposed scheme.

The rest of the paper is organized as follows. Section 2 briefly introduces generic particle filter (GPF) algorithm, followed by detailed description of the proposed state compression scheme and its architecture in Section 3. Section 4 presents a case study of the scheme in terms of filtering accuracy and memory overhead and Section 5 concludes the paper.

2. GPF ALGORITHM

In state estimation problems, system state evolving with time can be modeled using the following two equations

$$X_t = f_t(X_{t-1}, u_t) \quad (1)$$

$$Z_t = g_t(X_t, v_t) \quad (2)$$

where t is time-step index, X_t, Z_t is the system state vector and observation vector, respectively. f_t, g_t is state transition function and observation function with u_t, v_t as additive process noise and observation noise. Analytical form of the two functions is assumed known in the model. The goal is to estimate the state vector $X_t, \forall t \in \mathbb{N}$ recursively from input observations $Z_{1:t} = \{Z_1, Z_2, \cdots, Z_t\}$.

Centralized implementation of generic particle filtering is the cyclical process of sampling, weight calculation and resampling applied to a set of random samples which are recursively updated with time to approximate the PDF $p(X_t|Z_{1:t})$. In sampling, a total number of N particles, with

This research was supported by Zhejiang Provincial Natural Science Foundation of China (No. LQ12F04002), National Natural Science Foundation of China (No. 61204030) and Zhejiang Provincial Information Processing and Automation Technology Key Discipline Open Foundation of China.

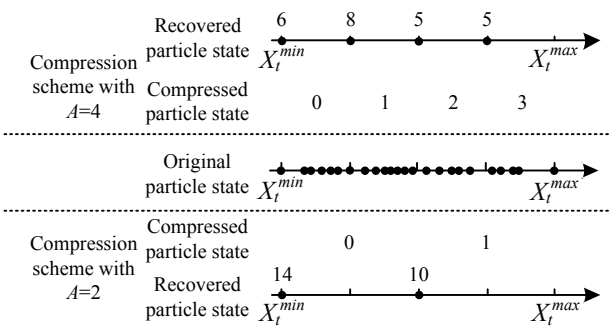

Fig. 1. Example of the compression scheme

each one represented by $X_t^{(n)}$ ($1 \le n \le N$), are drawn from an importance density $\pi(X_t)$, namely $X_t^{(n)} \sim \pi(X_t|X_{t-1}^{(n)}, Z_{1:t})$. Then in importance step, the weight of each particle is evaluated by a given observation using equation (3) and normalized by equation (4).

$$\omega_t^{*(n)} = \omega_{t-1}^{*(n)} \frac{p\left(Z_t|X_t^{(n)}\right)p\left(X_t^{(n)}|X_{t-1}^{(n)}\right)}{\pi(X_t^{(n)}|X_{1:t-1}^{(n)}, Z_{1:t})}, \quad (3)$$

$$\omega_t^{(n)} = \frac{\omega_t^{*(n)}}{\sum_{n=1}^{N} \omega_t^{*(n)}}. \quad (4)$$

The estimated result is obtained by

$$\hat{X}_t = \sum_{n=1}^{N} X_t^{(n)} \omega_t^{(n)}. \quad (5)$$

Finally, in resampling, particles with dominating weights are replicated and particles with negligible weights are discarded while keeping the number of particle set the same.

3. PROPOSED SCHEME AND ARCHITECTURE

In this section, proposed particle state compression scheme is introduced in detail and the resulting trade-off is analyzed. Architecture supporting the scheme is then presented including a generalized memory requirement discussion.

3.1. Proposed scheme

In a D-dimensional system, the interval of particle state values in each dimension is firstly divided into A sub-intervals after the particle set is sampled. Then, compressed particle information is calculated and stored. Before sampling of next iteration, particle states are recovered to continue filtering process. Table 1 summarizes the mathematical operations involved in the compression-recovery process at time-step t where $X(d)_t^{max}$, $X(d)_t^{min}$, $\Delta X(d)_t$ represents maximum, minimum and sub-interval length of the d^{th}-dimension and $X(d)_t^{(n)}$, $X(dc)_t^{(n)}$, $X(dr)_t^{(n)}$ denotes the d^{th} component of original, compressed and recovered n^{th} particle state vector, respectively ($1 \le d \le D$).

Figure 1 is an example of compression scheme applied to a one-dimensional state vector with 24 particles where A=2 and 4 are both shown. As can be seen, every particle state is converted into an integer ranging from 0 to A-1 and all particles in the same sub-interval range are compressed into the same one. As a result, the bit-width of compressed state is determined by parameter A regardless of the width of original states. Memory consumption then can be reduced when A is chosen properly that compressed particle states have shorter bit-width. In recovery, particles with the same compressed value turn into the same ones and the digits above the recovered states in Figure 1 indicate the number of particles concentrate on the corresponding state.

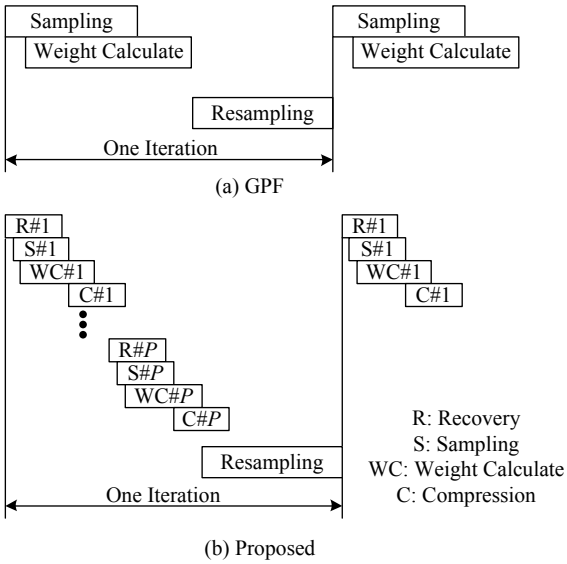

Fig. 2. Timing of GPF and proposed scheme

But, compression done after all particles are generated helps little in cutting down memory overhead since tremendous memory is already consumed to record the original uncompressed particle states. This issue is solved by employing the technique that particles are processed in even sub-groups sequentially. Once a sub-group of particles are sampled, compression is applied, producing and storing the compressed state values. Sampling is completed by repeating similar process for all sub-groups. Since the scheme is totally independent from weight variables, resampling is not influenced. The modified timing flow is shown and compared to GPF in Figure 2 where P denotes the number of sub-groups in proposed scheme. In this way, the size of memory for original states is reduced to one sub-group and can be used cyclically for different sub-groups. In addition, minimum and sub-interval length values of sub-groups are all stored since they may vary among sub-groups.

Table 1. Mathematical operations in the scheme

Calculating sub-interval length in each dimension
$$\Delta X(d)_t = \frac{X(d)_t^{max} - X(d)_t^{min}}{A}$$
Compression: for every particle in each dimension
$$X(dc)_t^{(n)} = floor\left(\frac{X(d)_t^{(n)} - X(d)_t^{min}}{\Delta X(d)_t}\right)$$
Recovery: for every particle in each dimension
$$X(dr)_t^{(n)} = X(d)_t^{min} + X(dc)_t^{(n)} * \Delta X(d)_t$$

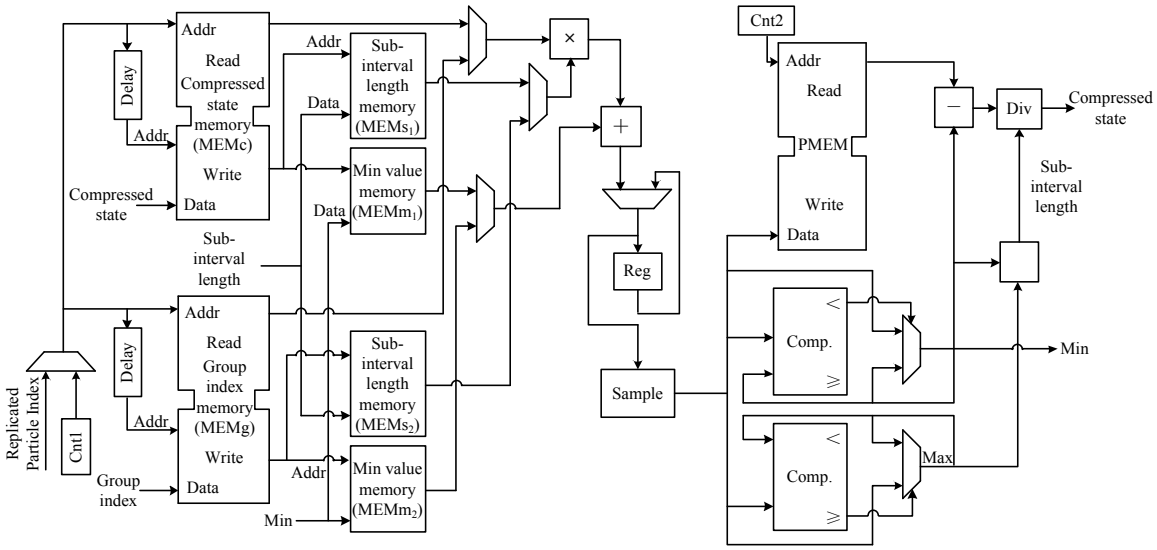

Fig. 3. Sampling architecture of proposed scheme

Group index of a particle is utilized, defining which sub-group it belongs to and values are chosen accordingly in recovery to guarantee the correctness of recovered states.

Since particles are processed in original bit-width and resampling is done on the whole particle set, the process of the scheme differs little from classical particle filtering algorithm except for the fact that recovered particle states for sampling of next iteration get shifted when compared to original ones. Consequently, deviations in each dimension are introduced as the trade-off for memory reduction. Considering that deviations may not necessarily at the same order of magnitude among dimensions, they are treated independently and equation (6) defines *dev* in d^{th}-dimension ($1 \le d \le D$) the way similar to quantization error.

$$dev(d) = \frac{\sum_{n=1}^{N}\left(X(d)_t^{(n)} - X(dr)_t^{(n)}\right)^2}{N-1} \quad (6)$$

Also noticed in Figure 1, when A increases, part of recovered states will be more close to original ones while remaining states kept the same. This yields a reversely proportional relation between A and *dev*. The scheme will then be just the same with GPF for $A \to \infty$. Therefore, threshold value A_T exists that for $A \ge A_T$, *dev* is negligible and the scheme can be treated equivalent to that of GPF, which ensures filtering performance of the scheme. This issue is further discussed in the case study in Section 4.

3.2. Architecture

Figure 3 presents the proposed sampling architecture for the scheme while remaining weight calculation and resampling architecture are kept the same with GPF. Considering various resampling techniques, Systematic Resampling (SR) [4] is chosen as the outputs are index of replicated particles and its replication factor.

After a particle is sampled, it is used to update the minimum or maximum value as well as stored in PMEM for the compression followed. When a sub-group of particles are sampled, sub-interval length of the sub-group in each dimension is calculated, kicking off the process of compression. Compressed states are generated with subtraction-division and stored in MEMc. The corresponding minimum, sub-interval length and group index are stored in $MEMm_i$, $MEMs_i$ (i=1,2) and MEMg for recovery, respectively. When all sub-groups are processed, resampling starts. Replicated particle index, generated by resampling, is used to access the compressed states and the corresponding minimum and sub-interval length. By a multiplication and addition, one particle state is recovered to continue sampling in next recursion. Ping-Pong scheme is supported here as pairs of MEMm and MEMs are used.

Since only simple logic units are introduced while the rest parts including resampling and weight calculation keep the same, the complexity of whole system remains not much influenced. And by adopting the architecture, memory requirement for storing particle states is analyzed and compared to GPF. Table 2 summarizes the comparison where N, D, k denotes the number of particles used, system dimension and bit-width of original particle state in each dimension. P and A have the same assignment as before. The components in the expression of proposed scheme correspond to PMEM, $MEMm_i$ and $MEMs_i$ (i=1,2), MEMc, MEMg in Figure 3, respectively.

Table 2. Memory requirement comparison

	Memory for storing particle states
GPF	kDN
Proposed	$kDN/P + 2kDP + ND\log_2 A + N\log_2 P$

For a given application where N, D, k is defined and by treating P as the variable, memory requirement is minimized when equation (7) is satisfied. A is independent from P and its influence is discussed in Section 4.

$$P = \frac{\sqrt{\left(\frac{N}{ln2}\right)^2 + 8k^2D^2N} - \frac{N}{ln2}}{4kD} \quad (7)$$

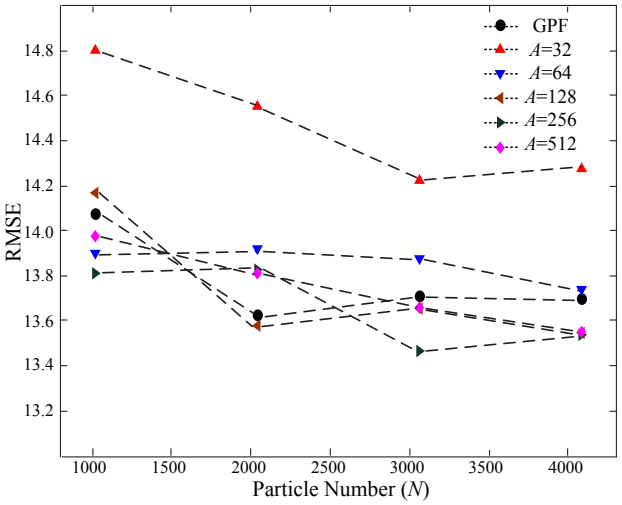

Fig. 4. RMSE results of GPF and proposed scheme

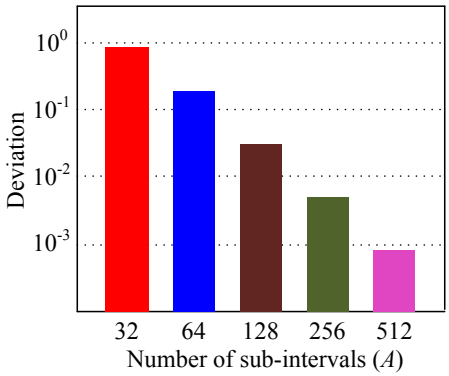

Fig. 5. Deviation example

4. EVALUATION

A case study for the proposed scheme in terms of filtering performance and particle state memory consumption is presented in this section under the 5-dimensional Nearly Constant Turn (NCT) [10] system, where k=16 in classical realization.

4.1. Filtering performance

Simulations are conducted with Root Mean Square Error (RMSE) as criterion for both GPF and the proposed scheme. Simulated cases are particle number N=1024, 2048, 3072, 4096 and by utilizing equation (7), P= 18.5, 24.1, 27.7, 30.4 for different N. To guarantee an integer value for number of particles in a sub-group, P is chosen as 16 for simplicity in all cases in proposed scheme. Moreover, different A values are chosen to illustrate its influence on filtering performance. Figure 4 plots the simulated RMSE while the deviation in the first dimension of state vector at N=2048 is depicted in Figure 5 as an example.

As can be seen from Figure 4, RMSE keeps at the same level with respect to GPF for $A \geq 64$. And obviously, deviation drops with increasing A as Figure 5 demonstrates.

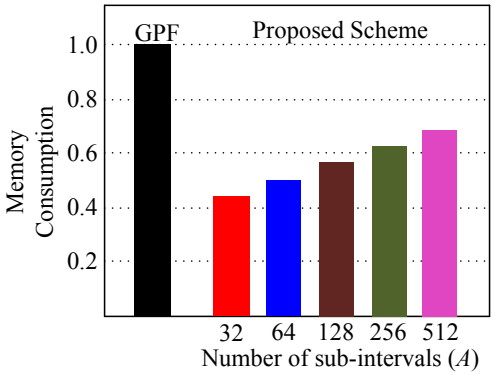

Fig. 6. Particle states memory consumption

By combining previous analysis, the threshold value A_T=64 in this case as deviation introduced by the compression-recovery scheme is negligible and the process of the proposed scheme can be treated equivalent to that of GPF since same level of satisfying filtering performance is achieved. For $A<A_T$, accuracy tends to be poorer since recovered particle states are more concentrated on limited values. The resulting less diverse particle set will deteriorate the estimation precision.

4.2. Particle state memory consumption

Particle state memory consumption is the primary concern of the proposed scheme and in this case, a more direct clarification other than the expression in Table 2 is provided in Figure 6, where memory consumption in simulated case of N=2048, P=16, k=16 are compared, with GPF normalized to 1. Memory consumption of proposed scheme grows linearly when A increases exponentially, which is easily seen from the expression in Table 2.

Although minimum memory requirement for storing particle states is achieved at A=32, but memory reduction without sacrificing filtering accuracy is acceptable, i.e., $A \geq A_T$. Minimum requirement is therefore attained at A=64, 50.31% of GPF and the corresponding bit-width of compressed states is $\log_2 64$=6. In practical implementation with N, D, k already defined, memory overhead can be further reduced since P is not an optimized value for all cases in the simulation.

5. CONCLUSIOIN

Presented in this paper is the particle state compression scheme and its architecture for centralized implementation. Memory requirement for storing particle states can be eased in the scheme with the trade-off of deviations introduced by the compression-recovery process. In the case study under NCT scenario, results show that proposed scheme is equivalent to GPF for A no less than 64 since filtering accuracy is maintained. Memory consumption for storing particle states can be reduced up to 49.69% when compared to traditional realizations.

REFERENCES

[1] N.J. Gordon, D.J. Salmond, A.F.M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEEE Proc. of Radar and Signal Processing*, vol. 140, pp. 107-113, 1993.

[2] F. Gustafsson, F. Gunnarsson, N. Bergman, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. on Signal Processing*, vol. 50, issue 2, pp. 425-437, 2002.

[3] M. Bolic, S. Hong, P.M. Djuric, "Finite precision effect on performance and complexity of particle filters for bearing-only tracking," *Conference Record of Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 838-842, 2002.

[4] A. Athalye, M. Bolic, S. Hong, "Generic Hardware Architectures for Sampling and Resampling in Particle Filters," *EURASIP Journal on Applied Signal Processing*, vol. 17, pp. 2888-2902, 2005.

[5] Q. Cheng, P. Bondon, "An Efficient Two-Stage Sampling Method in Particle Filter," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 48, issue 3, pp. 2666-2672, 2012.

[6] X. Fu, Y. Jia, "An Improvement on Resampling Algorithm of Particle Filters," *IEEE Trans. on Signal Processing*, vol. 58, issue 10, pp. 5414-5420, 2010.

[7] A.C. Sankaranarayanan, A. Srivastava, R. Chellappa, "Algorithmic and Architectural Optimizations for Computationally Efficient Particle Filtering," *IEEE Trans. on Image Processing*, vol. 17, issue 5, pp. 737-748, 2008.

[8] N. Zheng, Y. Pan, X. Yan, "Local weight mean comparison scheme and architecture for high-speed particle filters," Electronic Letters, vol. 47, issue 2, pp. 142-144, 2011.

[9] H.A.A. El-Halym, I.I. Mahmoud, S.E. Habib, "Efficient hardware architecture for Particle Filter based object tracking," *IEEE ICIP*, pp. 4497-4500, 2010.

[10] A.S. Bashi, V.P. Jilkov, X.R. Li, "Distributed implementation of particle filters," *Proc. of International Conference of Information Fusion*, vol. 2, pp. 1164-1171, 2003.