J Sign Process Syst (2014) 75:191–202
DOI 10.1007/s11265-013-0798-3

Efficient Uniform Quantization Likelihood Evaluation for Particle Filters in Embedded Implementations

Qifeng Gan · J. M. Pierre Langlois · Yvon Savaria

Received: 29 October 2012 / Accepted: 31 May 2013 / Published online: 30 June 2013

Abstract In this paper, we propose a uniform quantization likelihood evaluation (UQLE) algorithm for particle filters (PFs). This algorithm simplifies the exact likelihood evaluation (ELE) algorithm, the most computationally demanding function in PFs, by using a uniform quantization scheme to generate approximated weights. Simulation results indicate that PFs using UQLE can achieve comparable or better accuracy than PFs using ELE. The software implementation of UQLE for the bearing-only tracking (BOT) model in fixed-point arithmetic with 32 quantized intervals achieves 39.5× average speedup over the software implementation of ELE. An Application-specific Instruction-set Processor instruction was designed to accelerate the UQLE algorithm in a hardware implementation. The custom instruction implementation of UQLE for the BOT model with 32 intervals achieves 23.0× average speedup over the software implementation on a general-purpose processor with 5 % additional gates.

Keywords Particle filters · Likelihood evaluation · Embedded implementation · Application-specific instruction-set processor

1 Introduction

Particle filters (PFs) [1–4] are statistical signal processing methods that perform sequential Monte Carlo estimation based on a particle representation of probability densities. Since their introduction in 1993 [5, 6], PFs have gained in popularity to solve non-linear and/or non-Gaussian applications. They have shown great promise as a powerful methodology in addressing a wide range of complex applications including visual tracking [7, 8] and navigation [9, 10]. PFs use the concept of importance sampling to recursively compute the relevant probability distributions conditioned on the observations. In comparison with the Extended Kalman Filter (EKF) [11, 12], PFs do not rely on linearization techniques and can robustly approximate the true system state with an appropriate number of particles. In contrast, the Extended Kalman Filter sometimes has poor performance, lacks robustness and may introduce large biases [5].

One of the drawbacks of PFs comes from their significant computational requirements. This feature tends to limit their use in some embedded applications requiring real-time, high throughput processing. There are two main reasons for the significant computational requirements in PFs. The first reason is that a large number of particles are often required in order to achieve acceptable accuracy. As the number of particles increases, the processing speed of the PF tends to be seriously affected, although this problem can be mitigated through the use of a distributed architecture [13]. The second reason is related to the type of operations involved. In addition to non-linear operations in the dynamic state space (DSS) model defined by the target applications, traditional PFs, such as Sample-Importance-Resampling (SIR) PFs, may require non-linear operations such as division and exponentiation to calculate the particle weights in the likelihood evaluation step. These expensive and complex operations are often important bottlenecks in embedded implementations of PFs. Simplifying such complex operations is therefore a promising first step in order to improve the PF processing speed and energy efficiency.

PFs are commonly implemented in General-Purpose Processors (GPP) or Digital Signal Processors (DSP). Bolic used a TI TMS320C54x DSP to implement SIR PFs as a reference [14]. In that DSP, the exponential operation is approximated by a Taylor series. Implementation in GPPs and DSPs is

Q. Gan (✉) · J. M. P. Langlois
Department of Computer and Software Engineering, Polytechnique Montréal, QC H3C 3A7, Montreal, Canada
e-mail: Qifeng.gan@polymtl.ca

Y. Savaria
Department of Electrical Engineering, Polytechnique Montréal, QC H3C 3A7, Montreal, Canada

advantageous from a programmability point of view, but may not satisfy the performance requirements of applications that demand very high throughput. A hardware implementation is an appealing solution to this problem. This can include custom processors in Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). Several works have reported PF designs based on this approach [14–18]. These studies focused on doing the parallelization on the resampling step, which is a sequential portion of the algorithm. Hendeby et al. implemented SIR PFs in a Graphics Processing Unit (GPU) [19]. The speed increase was significant due to the nature of parallelization in GPU and the interpolation method for the exponential operation, but this solution is not suitable in general for embedded applications.

A hardware implementation can provide higher performance, but at the expense of making the implementation less flexible. A modification to an application's specifications may require significant redesign effort. Consequently, the other class of hardware implementation considered in this paper is the Application-Specific Instruction-set Processor (ASIP) [20]. ASIPs aim to strike a balance between GPPs and custom processors by combining a programmable solution with customized hardware units. With this approach, once the bottlenecks of the PFs are found, local acceleration can be applied with customized hardware units to improve the application's overall performance.

In this paper, we target SIR PFs, where resampling processing is a necessary step. We analyze their characteristics and profile them to identify possible bottlenecks for three applications. We specifically consider the likelihood evaluation. Previous work only focused on the exponential operation of the likelihood evaluation step. We broaden the scope of simplification. We also present a customized hardware unit for the proposed simplified algorithm, which can be locally accelerated with an ASIP. The main contributions of the paper are:

1) A characterization of PFs, which, unlike previous work [9, 14], focuses on distinguishing the application-specific blocks and the algorithm-defined blocks.
2) A demonstration that the likelihood evaluation is a significant and sometimes dominant step affecting PF throughput, and that it is worth optimizing.
3) A novel efficient uniform quantization likelihood evaluation (UQLE) algorithm to replace the exact likelihood evaluation (ELE) algorithm.
4) An evaluation of the impact of the approximated weights generated by UQLE under various options on the accuracy of the target applications.
5) An efficient customized instruction for UQLE that achieves significant local acceleration in an ASIP.

The rest of the paper is organized as follows. In Section 2, we describe the framework of PFs, analyze the computational properties of SIR PFs and justify the need for UQLE. The proposed UQLE is presented in detail in Section 3. In Section 4, we evaluate the accuracy of the proposed UQLE in comparison with that of ELE and profile both approaches in a general-purpose processor. In Section 5, we evaluate the throughput of UQLE in its software and customized ASIP versions. Section 6 concludes the paper.

2 Computational Characteristics of PFs

In this section, we briefly introduce PFs and analyze the computational complexity of SIR PFs to justify the need for the simplification of the likelihood evaluation.

2.1 Description of DSS Models and Examples

PFs are applied to problems that can be written in the form of DSS models. Consider the general form of a DSS model expressed by Eqs. (1) and (2):

$$\mathrm{x_t} = \mathrm{f_{t-1}}(\mathrm{x_{t-1}}, \mathrm{u_{t-1}}), \tag{1}$$

$$\mathrm{z_t} = \mathrm{h_t}(\mathrm{x_t}, \mathrm{w_t}), \tag{2}$$

where $\mathrm{t} \in \mathbb{N}$ is the time index, $\mathrm{x_t}$ is the target state vector, and $\mathrm{z_t}$ is the observation vector. $\mathrm{f_{t-1}}$ and $\mathrm{h_t}$ are possibly nonlinear prediction functions of the state $\mathrm{x_{t-1}}$ and observation functions of the state $\mathrm{x_t}$, respectively. $\mathrm{u_{t-1}}$ is the process noise sequence and $\mathrm{w_t}$ is the observation noise sequence. Equation (1) describes how the state vector $\mathrm{x_t}$ evolves with time. Equation (2) develops the noisy observation vector $\mathrm{z_t}$ as a function of the state vector $\mathrm{x_t}$. In the PF framework, the aim is to learn about the unobserved state based on a set of noisy observations as time evolves.

In this paper, we apply DSS models to three concrete examples: Linear Gaussian (LG), Uni-variate Nonstationary Growth (UNG) and Bearing-only Tracking (BOT) models.

2.1.1 Example 1: LG Model

Consider the following LG model [1]:

$$x_t = x_{t-1} + u_t, \tag{3}$$

$$z_t = x_t + w_t, \tag{4}$$

where $\mathrm{u_t} \sim \mathcal{N}(0, \sigma_\mathrm{u}^2)$ and $\mathrm{w_t} \sim \mathcal{N}(0, \sigma_\mathrm{w}^2)$, and where σ_u^2 and σ_w^2 are considered fixed and known with variance $\sigma_\mathrm{u}^2 = \sigma_\mathrm{w}^2 = 1$. The initial state distribution is $\mathrm{x_0} \sim \mathcal{N}(5, 1)$. This is the basic model for estimating the problems.

2.1.2 Example 2: UNG Model

We consider here a classical UNG model which has been used extensively in the literature for benchmarking numerical filtering techniques [1, 2, 5]. The state space equations are as follows:

$$x_t = \frac{x_{t-1}}{2} + 25\frac{x_{t-1}}{1+x_{t-1}^2} + 8\cos(1.2t) + u_t, \tag{5}$$

$$z_t = \frac{x_t^2}{20} + w_t, \tag{6}$$

where $u_t \sim \mathcal{N}(0, \sigma_u^2)$ and $w_t \sim \mathcal{N}(0, \sigma_w^2)$, with $\sigma_u^2 = 10$ and $\sigma_w^2 = 1$. The initial state distribution is $x_0 \sim \mathcal{N}(0.1, 5)$.

We choose this model because it is highly nonlinear and its observation Eq. (6) introduces an interesting bimodal problem to the estimation. This makes the estimation problem more difficult to solve by traditional methods such as the Kalman filter.

2.1.3 Example 3: BOT Model

This example is placed in the context of radar-based target tracking. The BOT model concerns an object moving in the x-y plane (2-D space). The observations taken by the sensor to track the object are in terms of the bearing or angle with respect to the sensor [5, 14].

Let the sensor be stationary and located at the origin in the x-y plane. The object moves according to the following state space model:

$$X_t = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} X_{t-1} + \begin{bmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_{x_t} & u_{y_t} \end{bmatrix}^T \tag{7}$$

$$z_t = \tan^{-1}\left(\frac{y_t}{x_t}\right) + w_t \tag{8}$$

where $X_t = \begin{bmatrix} x_t & v_{x_t} & y_t & v_{y_t} \end{bmatrix}^T$, x_t and y_t denote the coordinate position of the target and v_{x_t} and v_{y_t} denote the velocities in the x and y directions, respectively. The vector $\begin{bmatrix} u_{x_t} & u_{y_t} \end{bmatrix}^T$ is composed of white Gaussian noise variable with standard derivation σ_u=0.001. Parameter $w_t \sim \mathcal{N}(0, \sigma_w^2)$ is the observation noise with σ_w=0.005. The set of initial states is set to $X_0=[-0.05, 0.001, 0.7, -0.055]^T$ and σ_1= 0.5, σ_2=0.005, σ_3=0.3 and σ_4=0.01, which are the standard derivations of the noise for the initial state.

From Eq. (8), we see that no range information is available to the sensor. Thus, only the angle of the object movement but not its distance from the point of sensor can be measured with a series of observations. The observation consists of a modal ridge along the line $y_t = \tan(z_t)\, x_t$. Hence, the BOT model is the multimodal case. The estimate of the object trajectory is difficult to solve because it only depends on this multimodal measurement and the prior information, which is the position and the velocity of the object at the initial stage.

2.2 Overview of the Particle Filtering Framework

PFs base their operations on approximating the a posteriori probability density function (PDF) via a set of N weighted samples $\{x_{0:t}^i, \omega_t^i\}_{i=1}^N$ called particles. These particles are drawn independently from an importance density $q(x_t|x_{t-1}, z_{1:t})$. Accordingly, if the observation $z_{1:t}=\{z_j, j=1,\ldots,t\}$ is available and the weights are normalized such that $\sum_i \omega_t^i=1$, the a posteriori PDF at time t can be approximated as:

$$p(x_{0:t}|z_{1:t}) \approx \sum_{i=1}^N \omega_t^i \delta\left(x_{0:t} - x_{0:t}^i\right) \tag{9}$$

where $x_{0:t}=\{x_j, j=0,\ldots,t\}$ is the set of all states up to time t.

After knowing the a posteriori PDF $p(x_{0:t}|z_{1:t})$, the state estimate x_t can be computed using either the minimum mean-square error (MMSE), the maximum a posteriori probability (MAP) or other methods.

In the implementation of PFs, the a posteriori PDF $p(x_{0:t}|z_{1:t})$ may be obtained recursively through the following three basic steps:

1) Prediction generation: the particle x_t^i is drawn by the importance density $q(x_t|x_{t-1}, z_{1:t})$. The choice of the importance density plays a fundamental role in the design of particle filters because generating the particles and the particle weights is related to this important density [3]. A standard scheme is to choose the state transition a priori probability $p(x_t|x_{t-1})$ defined by (1) as the importance density. The advantage of this scheme is that it is efficient to implement.
2) Weight calculation: when the likelihood distribution $p(z_t|x_t)$ is obtained upon the arrival of the observation z_t, the particle weights can be computed via the weight update equation as follows:

$$\omega_t^i = \omega_{t-1}^i \frac{p\left(z_t|x_t^i\right) p\left(x_t^i|x_{t-1}^i\right)}{q\left(x_t^i|x_{t-1}^i, z_{1:t}\right)}. \tag{10}$$

Normalization is carried out as follows:

$$\omega_{n_t}^{\ i} = \frac{\omega_t^i}{\sum_{i=1}^N \omega_t^i}. \tag{11}$$

3) Resampling processing: new particles $\widetilde{x}_t^i$ are drawn from the set of particles x_t^i based on the particle weights ω_t^i through a resampling scheme. The resampled set of new particles and their weights is denoted by $\{\widetilde{x}_t^i, \widetilde{\omega}_t^i\}$.

There are many variations of PFs [1, 2]. In this paper, the analysis of PFs is related to SIR PFs, whose detailed pseudo-code is shown in Fig. 1. After initialization, the algorithm iterates through five steps: prediction generation, weight calculation, weight normalization, resampling, and estimate calculation. The weight normalization step can be merged with the resampling and estimation calculation steps. Such merging can reduce the execution time of PFs.

2.3 Computational complexity of SIR PFs

2.3.1 Functional view of SIR PFs

The functional blocks of SIR PFs with merged steps (weight normalization resampling, and estimation) are shown in Fig. 2. For each observed input, the SIR PFs sequentially perform three steps: prediction generation, weight calculation and resampling processing. Prediction generation is decomposed into two blocks, and weight calculation is decomposed into three blocks. Resampling processing is made up of a single block. The three main steps, decomposed into a total of six blocks, are in the critical path. Thus, all these blocks are potential optimization targets when a high performance implementation is required. The complexity of SIR PFs can be partitioned into the six considered blocks: Transition Processing (TP), Random Number Addition (RNA), Particle Measurement Processing (PMP), Distance Calculation (DC), Likelihood Evaluation (LE) and Resampling Algorithm (RA). The TP and PMP blocks are application-specific blocks and their respective complexity depends on the application. The other blocks are algorithm-defined. Their complexity is closely related to the selected algorithms. For example, the normal or exponential likelihood evaluation can be used in the LE block. The choice of the algorithms depends on the type of operations that must be used to obtain a suitable accuracy, a suitable complexity or an appropriate trade-off between them.

There are two additional significant blocks in Fig. 1. One calculates the estimates for the filter output. The other generates random numbers used for updating the states in the prediction generation step. Random number generation can consume a substantial portion of the overall processing time. But due to the fact that it is not in the critical path, a random number generator implemented as a co-processor can be employed to operate in parallel and not affect the speed of PFs. Similarly, the output block is not in the critical loop.

```
a) Initialization: Generate $x_0^i \sim p\ x_0$   i =1……. N.
b) Prediction generation: $x_t^i \sim p(x_t|x_{t-1}^i)$
c) Weight calculation: $\omega_t^i = \omega_{t-1}^i p\ z_t\ x_t^i$ .
d) Weight normalization: $\omega_{n_t}^i = \frac{\omega_t^i}{{}_{i=1}^{N} \omega_t^i}$.
e) Resampling processing:
   $x_t^i, \omega_t^i \}_{i=1}^N = RESAMPLE\ \{x_t^i, \omega_{n_t}^i\}_{i=1}^N$ .
f) MMSE Estimate calculation $x_t = {}_{i=1}^{N} \omega_t^i x_t^i$.
g) Let $t = t + 1$ and repeat from (b) when observation is available.
```

Figure 1 Pseudo-code of SIR PFs.

2.3.2 Computational complexity

The computational complexity of SIR PFs depends on the complexity and dimensionality of the underlying DSS model. Indeed, that model is updated in the TP and PMP blocks. Most of the computational effort of the PFs may be spent in these two blocks if the DSS model is very complex or has a large number of dimensions. From Table 1, we can see that the TP block in the UNG model and the PMP block in the BOT model require 51.5 % and 45.4 % of the overall execution time in a GPP without floating-point unit (FPU), respectively. It is in general not possible to analyze and determine the computational property of the DSS model without knowing the specific application. In contrast, with the LG model, the TP and PMP blocks require only about 0.1 % of the whole execution time. In this paper, we focus on the analysis of the generic blocks that compose the particle filtering algorithm. Because the TP and PMP blocks are application specific blocks, we concentrate our efforts on analyzing the other blocks that are in its critical path: RNA, DC, LE, and RA.

Tables 1 and 2 present profiling results for an Xtensa LX2 processor [21] with and without FPU, respectively. The tables include average cycle counts and percentage of total execution time consumed by each block for the LG, UNG, and BOT models, after 50 runs. The Xtensa LX2 processor is a general-purpose processor with an optional FPU, some configurable units and that supports optional customizable instructions that can be added to improve performance.

From Table 1, we observe that the LE block always takes a significant portion of the total number of clock cycles (90 %, 41 %, and 47 % for the LG, UNG, and BOT models, respectively). Table 2 shows that when a FPU is employed, the LE block consumes a larger fraction of the total execution time, and the proportions increase to 98 %, 43 %, and 49 % for the LG, UNG, and BOT models, respectively. This is remarkable and may appear to be somewhat counterintuitive. Indeed, it was found that the FPU provided by the Xtensa LX2 processor does not support the complex operations (exponentiation, division, conditional branch, and so on) that the LE block often uses. The additional hardware complexity introduced by the FPU option is not very useful for the LE block, which is either dominant or very significant, as it does not improve

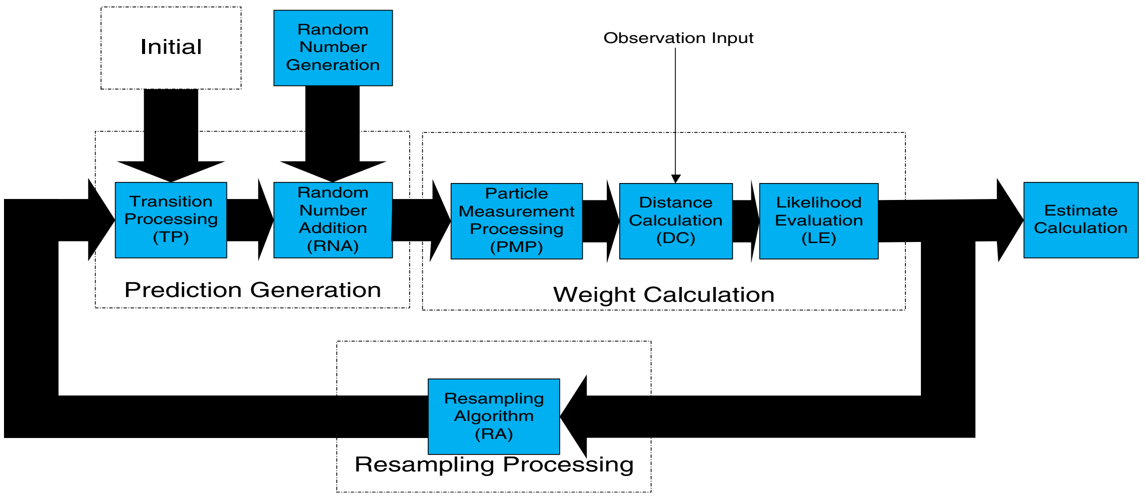

Initial
Random Number Generation
Observation Input
Transition Processing (TP)
Random Number Addition (RNA)
Particle Measurement Processing (PMP)
Distance Calculation (DC)
Likelihood Evaluation (LE)
Estimate Calculation
Prediction Generation
Weight Calculation
Resampling Algorithm (RA)
Resampling Processing

Figure 2 Functional view of SIR PFs.

much its performance. According to the results in Tables 1 and 2, when a GPP is used, whether a FPU is present or not, any attempt at optimizing SIR PFs must consider the LE block.

One popular way to reduce the computation time is to use a parallel implementation with multiple processing elements (PEs) or multi-cores. The execution time of all PF blocks, except the RA block, can be reduced significantly, because the computations are independent for each particle. In principle, the execution time in the LE block can be reduced by a factor equal to the number of PEs or cores. Due to its sequential nature, the RA block that performs systematic resampling (SR) [22], may become the most time consuming block when many PEs are employed. However, various efforts [14–16] have been made to derive distributed resampling algorithms/architectures which let the RA block partially run in PEs. The execution time of the RA block can also be reduced. When such distributed resampling algorithms/architectures are used, the LE block becomes the most computationally expensive part again. This is a further motivation to focus on optimizing the LE block, irrespective of whether PFs are executed in a single GPP or using a parallel implementation with multiple PEs.

3 Proposed UQLE Algorithm

Most likelihood evaluation algorithms used in PFs are based on the class of exponential family of distributions, which includes the normal, exponential, and gamma distributions. These distributions require the calculation of the exponentiation, division, multiplication and other operations. These operations make the likelihood evaluation become the most computationally intensive part in PFs. A simplified likelihood evaluation algorithm is one way to improve the particle filtering speed.

The proposed UQLE algorithm is based on the assumption that approximated weight values do not significantly affect the accuracy of the target application. Under this assumption, UQLE can employ a uniform quantization method to enable the use of a simple approximate representation for some quantity of the output value, the particle weights.

In SIR PFs, the particle weights can be given by the likelihood evaluation:

$$\omega_t^i = \exp\left(\frac{-d_i^2}{2{*}\sigma^2}\right), \qquad (12)$$

Table 1 Average cycle counts of LG, UNG, and BOT models using SIR PFs with 512 particles in the Xtensa LX2 processor without FPU.

	LG Model		UNG model		BOT model	
	Cycle counts (K)	%	Cycle counts (K)	%	Cycle counts (K)	%
Transition processing (TP)	2.01	0.11	2045.86	51.53	34.98	0.90
Random number addition (RNA)	18.38	1.04	18.18	0.46	63.96	1.64
Particle measurement processing (PMP)	2.01	0.11	111.25	2.80	1769.81	45.41
Distance calculation (DC)	56.40	3.19	66.38	1.67	112.04	2.88
Likelihood evaluation (LE)	1593.61	90.01	1639.41	41.29	1827.19	46.88
Systematic resampling algorithm (RA)	98.17	5.54	89.45	2.25	89.27	2.29
Total	1770.56	100	3970.53	100	3897.25	100

Table 2 Average cycle counts of LG, UNG, and BOT models using SIR PFs with 512 particles in the Xtensa LX2 processor with FPU.

	LG model		UNG model		BOT model	
	Cycle counts (K)	%	Cycle counts (K)	%	Cycle counts (K)	%
Transition processing (TP)	2.01	0.13	1996.68	53.68	7.02	0.20
Random number addition (RNA)	5.52	0.35	5.52	0.15	10.04	0.28
Particle measurement processing (PMP)	2.01	0.13	83.24	2.24	1767.56	49.77
Distance calculation (DC)	5.52	0.35	5.52	0.15	5.52	0.16
Likelihood evaluation (LE)	1552.50	97.55	1604.82	43.14	1734.22	48.85
Systematic resampling algorithm (RA)	23.72	1.49	23.77	0.64	26.09	0.74
Total	1591.27	100	3719.55	100	3550.45	100

where d_i, the input to the likelihood evaluation, is the ith particle distance between the observation and the ith particle result in the PMP block, and σ^2 is the variance of the observation. The proposed UQLE takes advantage of the fact that the particle weights are in the range [0, 1]. This range can be divided into M intervals of length Q, where Q is the quantization step-size:

$$Q = \frac{1}{M}. \tag{13}$$

Based on this splitting, the range of the weights consists of M intervals: $\{[0,Q),[Q,2Q),[2Q,3Q),\ldots,[(M-2)Q,\ (M-1)Q),[(M-1)Q,1]\}$. Inside each interval, we can choose the high value, the middle value or the low value to represent the quantized value as the approximated weight W_m. They are defined as:

$$W_m = \begin{cases} mQ, & \textit{low value} \\ mQ + \dfrac{Q}{2}, & \textit{middle value} \\ (m+1)Q, & \textit{high value} \end{cases}, \tag{14}$$

where $m=0,1,\ldots,M-1$, is the index of the intervals. With this approach, once the index of the intervals is determined according to the particle distance d_i, the approximated weights W_m are given by Eq. (14).

In accordance with the M intervals of the particle weights, the data range of the particle distance can also be divided into M intervals: $\{[T_1,+\infty),[T_2,T_1),[T_3,T_2),\ldots,[T_{M-1},T_{M-2}),[0,T_{M-1}]\}$, where $T_m, m=0,1,\ldots,M-1$ is the boundary of each interval in the particle distance direction. They can be obtained via the inverse likelihood evaluation. The inverse normal distribution likelihood evaluation is given by:

$$T_m = \sqrt{-\ln(mQ)*2*\sigma^2}. \tag{15}$$

When the particle distance d_i is available, the index of the interval m can be determined from the interval of the distance range in which this d_i falls.

When there are significant deviations from the most recent estimate, the value of all the weights can be very small, and the distances can all fall in the $[T_1,+\infty)$ range. This comes from the fact that all weights are equal to zero when we use the low value of the intervals as the approximated weight. In order to avoid this situation, we insert an additional interval $[0,\delta), 0<\delta\ll Q$ into $[0,Q)$. The approximated weight at the index $m=0$ needs to be modified as follows:

$$W_0 = \begin{cases} \delta, & d_i\in[T_\delta,+\infty) \\ 2\delta, & \textit{low value} \\ \dfrac{Q}{2}, & \textit{middle value} \\ Q, & \textit{high value} \end{cases}, \tag{16}$$

where T_δ is calculated by Eq. (15) with the input δ. There are therefore $M+1$ intervals.

Figure 3 shows an example of the likelihood evaluation curve for the proposed UQLE. The range of particle weights is divided into $M+1=5$ intervals: $\{[0,\delta),[\delta,Q),[Q,\ 2Q),[2Q,\ 3Q),[3Q,1]\}$. Then, T_δ, T_1, T_2, T_3 can be calculated by the inverse likelihood evaluation defined by Eq. (15). The index m can be determined via the comparison between the particle distance and T_δ, T_1, T_2, T_3. For instance, for a particle distance $d_i\in[T_3,T_2)$, we can obtain its index $m=2$. The weight for this particle distance can be approximated by Eqs. (14) and (16), which is equal to W_2.

The pseudo-code for UQLE is given in Fig. 4. The whole algorithm is divided into two parts: a pre-calculation part and an on-line execution part. After defining the number of intervals and the weight value, which is a low, middle or high value as the representation, the pre-calculation part provides the boundary in the distance direction T_m and the approximated weights for each interval according to Eqs. (14) and (16). These results are used for the on-line execution part later. In the on-line execution part, the approximated weights can be obtained by comparing the input, i.e. the particle distance, with the boundary in the distance direction T_m. Using the proposed procedure, the execution time is reduced significantly since we obtain the weights from comparisons instead of complex arithmetic for exact likelihood evaluation.

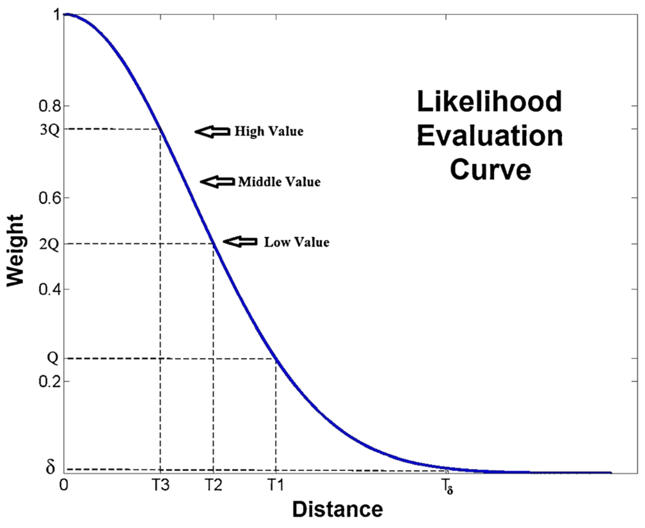


Figure 3 Likelihood evaluation curve for the proposed UQLE with $M+1=5$ intervals.

4 UQLE Accuracy Evaluation

4.1 Implementation Environment and Accuracy Metrics

In this section, we simulate SIR PFs using UQLE for the LG, UNG, and BOT models in the MATLAB environment. According to the UQLE algorithm, the particle distance needs to be compared with pre-calculation boundaries T_m. We can then find the index of the intervals m and obtain the particle weight directly from the pre-calculation weight W_m. The input and output do not have any numerical calculation relationships. It is not necessary to set all the data in UQLE in floating-point arithmetic. In order to be consistent with the implementation in the Xtensa processor, we implemented UQLE with floating-point arithmetic for the input, the particle distance, and fixed-point arithmetic with 16-bit representation for the output, the particle weight. Thus, we set the range [0, 65535] for the particle weight and δ equal to 1. Under this situation, we compare the resultant accuracy to SIR PFs using ELE. For the LG and UNG models, the accuracy is measured by the Root-Mean-Square Error (RMSE):

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(x_t - \widehat{x}_t\right)^2} \tag{17}$$

where x_t is the true simulated state and $\widehat{x}_t$ is the estimated state. For the BOT model, in order to obtain a better illustration for the performance, the accuracy is measured by the combined Mean-Squared-Error (MSE) (where MSE= RMSE^2) of the x and y positions:

$$MSE_{xy} = \sqrt{(MSE_X)^2 + (MSE_Y)^2} \tag{18}$$

where MSE_X and MSE_Y are the MSE for the x and y positions, respectively. The smaller the value of the combined MSE is, the better the performance we obtain. Furthermore, simulation for a lost track situation is conducted for the BOT model, as suggested in [14]. The track is considered lost if all particles have zero weight in the ELE or if all the particle weights are equal to δ in the UQLE.

In order to obtain stable performance results, 10,000 simulations were performed with 50 observation inputs for the LG and UNG models and 25 observations for the BOT model.

```
Pre-calculation part:
    Define $\delta,\ 0 < \delta \ll Q$
    $T_\delta = sqrt(-In\ \delta\ \ * 2 * \sigma^2)$.
    $T_m = sqrt\ \ -In\ mQ\ \ * 2 * \sigma^2\ \ , \mathrm{m} = 0,1,2,\ldots,\mathrm{M}-1$
On-line execution part:
$\omega_t^i = likelihood(d_t^i)$
      If $(d_t^i \geq T_\delta)$
                  $\omega_t^i = W_0$.
      else
                  Find  $m$, the index of the intervals,
            such that  $T_m > d_t^i \geq T_{m+1}$,
                  $\omega_t^i = W_m$.
      end
end
```

Figure 4 Pseudo-code of the proposed UQLE algorithm.

4.2 Accuracy for the LG Model

Table 3 shows average RMSE results for SIR PFs using ELE, for 16, 128, 512, 1,024, 2,048, and 4,096 particles. We use these results as the reference to compare the results generated by UQLE.

In Fig. 5, RMSE results of SIR PFs using UQLE are plotted. The dashed line gives RMSE results of SIR PFs using ELE as summarized in Table 3. From Fig. 5, we can see that even for a small number of particles N=16, SIR PFs using UQLE can produce RMSE performance close or equivalent to SIR PFs using ELE when M, the number of intervals, exceeds 32. Hence, UQLE can replace ELE in SIR PFs in the LG model.

It can also be observed that with the number of particles N increasing, UQLE with M=16 or M=8 can achieve equivalent accuracy to ELE. Decreasing the number of intervals implies reducing the computation time. UQLE outperforms ELE in speed without sacrificing the accuracy. Concerning the choice between the low, middle or high value, UQLE with the low value is slightly better than other choices except for the case of N=16 particles.

Table 3 Average RMSE of SIR PFs using ELE for the LG model.

# of particles	16	128	512	1,024	2,048	4,096
Average RMSE	0.854	0.791	0.785	0.784	0.783	0.783

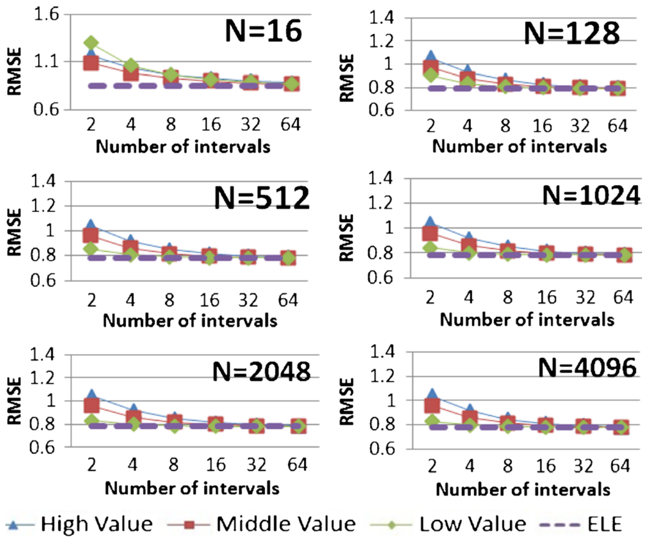


Figure 5 Average RMSE results of SIR PFs using UQLE for the LG model.

4.3 Accuracy for the UNG Model

Figure 6 compares the average RMSE results for 16, 128, 512, 1,024, 2,048 and 4,096 particles for the UNG model. The dashed line displays the reference accuracy of SIR PFs using ELE, with the values summarized in Table 4. Except for the case of N=16 particles, Fig. 6 shows that UQLE achieves an accuracy similar to ELE even for a small number of intervals (M=16) when the low or middle values are used. The results for the case of N=16 show that UQLE outperforms ELE in accuracy when the middle or high values are used with any number of intervals. In addition, the accuracy of UQLE in the UNG model is similar to that in the LG model, with UQLE using fewer intervals and achieving equivalent accuracy to ELE when the number of particle increases.

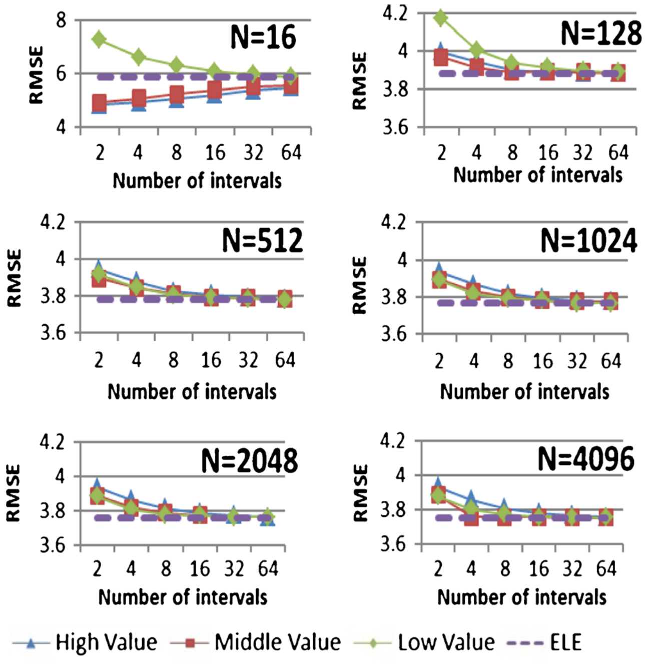


Figure 6 Average RMSE results of SIR PFs using UQLE for the UNG model.

Table 4 Average RMSE of SIR PFs with ELE for the UNG model.

# of particles	16	128	512	1,024	2,048	4,096
Average RMSE	5.85	3.88	3.78	3.76	3.76	3.76

4.4 Accuracy for the BOT Model

In Table 5, we show the average combined MSE and the number of times when the track is lost versus the number of particles for SIR PFs using ELE. From Table 5, with the increase of the number of particles, the combined MSE decreases from 0.27 for 128 particles to 0.14 for 4,096 particles. Meanwhile, the number of lost tracks drops from about 9,500 for 128 particles to below 2,700 for 4,096 particles. It should be noted that in ELE we assume that the weights are clamped to $\delta=2^{-16}$ when their value is smaller than 2^{-16}, which is the same as for UQLE, because the lost track situation significantly depends on the smallest weights considered.

Figure 7 shows the average combined MSE and the lost track situation for SIR PFs using UQLE. The dashed lines are the corresponding results obtained by SIR PFs using ELE. From Fig. 7, although UQLE employing the low value of the intervals has poor performance in our experiments, UQLE with the high value or the middle value of intervals significantly outperforms SIR PFs using ELE, even for the smallest number of intervals M=2 in our experiments. This means that the speed using the proposed UQLE can increase significantly in the BOT model because each particle only needs to be compared with two boundaries in the UQLE algorithm. Figure 8 displays a representative trajectory and the tracking obtained by SIR PFs using ELE and SIR PFs using UQLE with 2 intervals and high value. It shows again that UQLE with few intervals can replace ELE in the BOT model.

In summary, simulation results show that UQLE can achieve equivalent or better accuracy than ELE for SIR PFs when the number of intervals and the representation of the output value are suitably chosen. In our experiments, in order to obtain comparable accuracy to ELE, the minimum number of intervals is M=32, 16 and 2 for the LG, UNG, and BOT models, respectively.

Table 5 Average combined MSE and Lost track of SIR PFs with ELE for the BOT model.

# of particles	128	512	1,024	4,096
Average combined MSE	0.27	0.21	0.17	0.14
Lost track	9,503	8,052	6,824	2,637

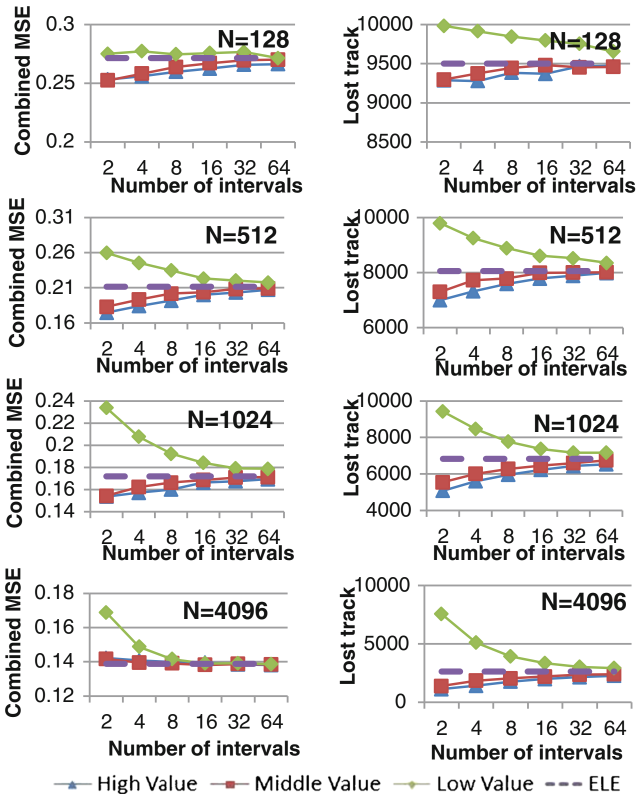


Figure 7 Average combined MSE and lost track for SIR PFs using UQLE.

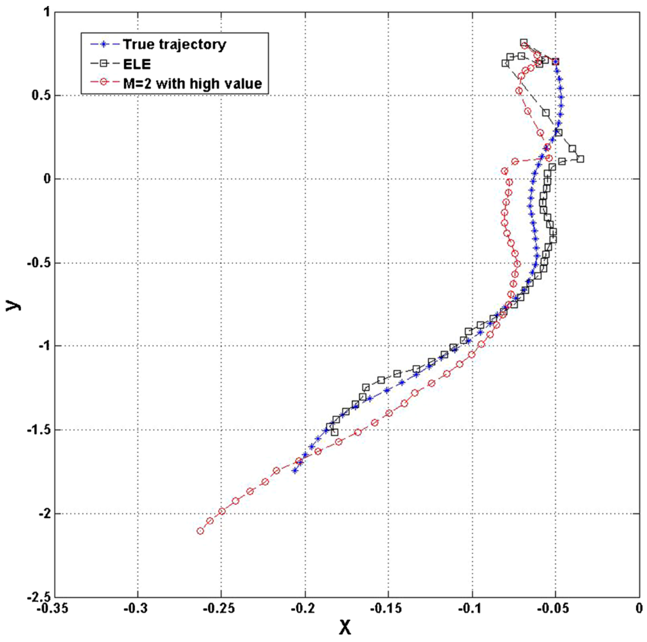


Figure 8 Tracking of a moving target in two dimensions with SIR PFs using ELE and SIR PFs using UQLE with 2 intervals and high value.

5 UQLE Throughput Evaluations

5.1 UQLE Software Implementation and its Performance

We implemented the proposed UQLE in the Xtensa processor in order to compare its speed performance to ELE's. We used the same configuration simulated in the MATLAB environment, with the particle distance expressed with floating point precision and the particle weights expressed in fixed point with 16 bits. In order to reduce the execution time, we still evaluate the performance when setting the particle distance in fixed-point arithmetic.

Table 6 shows the average speedup results for UQLE for the BOT model with the particle distance expressed with floating-point and fixed-point accuracy when compared to ELE. Adding a co-processor FPU in the Xtensa LX2 processor increases the processor size by approximately 50 K gates without improving performance, since the FPU does not support conditional branch instructions that are heavily used in the UQLE. The maximum execution time for the ELE in Table 1 is chosen as a baseline. UQLE using 64 intervals achieves 5.6× and 19.5× average speedup over ELE when executed in floating-point and fixed-point arithmetic for the particle distance, respectively. The speedup is greater when the number of intervals for UQLE is smaller than 64. UQLE with 4 intervals in fixed-point arithmetic can achieve 295.2× average speedup over the software implementation of ELE in a GPP.

As shown in Figs. 5, 6 and 7, in order to achieve the same accuracy as ELE, the UQLE requires 32, 16 and 2 intervals for the LG, UNG and BOT models, respectively. UQLE for the LG model is therefore the most computationally demanding of the three. Still, it achieves almost 40× speedup over the software implementation of ELE.

5.2 UQLE ASIP Implementation and Its Performance

We designed a custom instruction for UQLE to improve the speed of SIR PFs in ASIPs. As shown in Fig. 4, the UQLE

Table 6 Execution time of UQLE in PFs with 512 particles with different number of intervals in Xtensa LX2 processor.

Likelihood Evaluation (LE)	Particle distance representation	# of M	Average execution time (cycle counts)	Average speedup
ELE	Floating point	–	1827.19 K	1.0×
UQLE	Floating point	64	327.34 K	5.58×
		32	164.67 K	11.10×
		4	21.96 K	83.21×
	Fixed point	64	93.78 K	19.48×
		32	46.24 K	39.52×
		4	6.19 K	295.18×

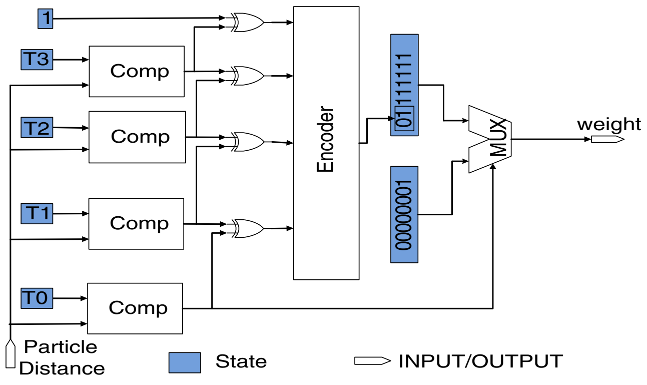


Figure 9 Custom UQLE logic organization with 4 intervals.

algorithm has one input d_t^i and one output ω_t^i. This situation is favorable for the generation of a custom instruction because there are no extra input/output data that could create a bottleneck preventing effective acceleration.

The proposed logic organization of the custom instruction is shown in Fig. 9. In a typical configuration of that instruction, 4 quantization intervals are supported and the output is the high value of the intervals expressed with 8 bits. The critical path consists of a comparator, an XOR gate, an encoder and a multiplexer. When a particle distance is available, this value is compared with distance boundaries $\{T_0,T_1,T_2,T_3\}$, which are stored in the state registers. For example, if $d\in[T_1,T_2]$, which means $d<T_0, d<T_1$, $d>T_2$ and $d>T_3$, the outputs of the comparators are {0,0,1,1}. Through the XOR gates, the encoder input is changed to {0,1,0,0}. At the output of the encoder, the interval index is equal to 01, which means d is in the second interval. The weight can be set to {01111111}, which corresponds to the high value of the second interval. The number of intervals supported is a parameter of the architecture that can easily be changed as a function of the specific requirements. We implemented the instruction for five numbers of intervals (4, 8, 16, 32 and 64) and three choices of W_m (low, middle and high value of the quantization intervals).

The custom instruction was described in the Tensilica Instruction Extension (TIE) language [23] to generate a specific functional unit in the Xtensa LX2 processor. The special instruction for UQLE is:

$$\omega_t^i = likelihood\left(d_t^i\right). \qquad (19)$$

Table 8 Speedup between running in GPP and in GPP with UQLE instruction for the LG, UNG, and BOT models.

Applications	Average execution time (GPP)	Average execution time (GPP with UQLE Instruction)	Average speedup
LG	1770.56 K	179.20 K	9.88×
UNG	3970.53 K	2333.37 K	1.70×
BOT	3897.25 K	2072.31 K	1.88×

Profiling results are shown in Table 7. With the custom instruction, the speedup can reach up to an average of 909.1× and 865.8× over the implementation of ELE in a GPP without FPU and with FPU, respectively. The maximum average cycle counts for ELE in our experiments are extracted to calculate this speedup. In addition, the cycle count for UQLE is independent of the number of intervals. Increasing the number of intervals only increases the processor size. The additional size for UQLE with 4, 32, 64 intervals, respectively only occupies 0.7 %, 4.7 %, 8.4 % of the reference processor that consumes 79 K gates.

From Table 7, the execution time can be significantly reduced from 1827.2 K cycles to only 2.0 K cycles on average when the UQLE instruction is used. For the LG model, the likelihood evaluation dominates the execution time of the whole application. It is significant that using only 3.75 K additional gates for UQLE instruction with 32 quantization intervals can achieve almost 10× speedup as shown in Table 8. These additional gates only occupy 4.7 % of the reference processor. If a processor with multiple PEs is used to implement the LG model, at least 10 PEs are required to achieve 10× average speedup. But the number of additional gates to build 10 PEs is much larger than the number of gates that the UQLE instruction uses. Hence, for those applications using PFs where the DSS model is not complex, using UQLE is a powerful first step to reduce the execution time and energy consumption.

For applications using PFs like the UNG and BOT models, where the DSS model is rather complex, simplifying the algorithm or using hardware implementation must be shifted to the DSS model. Finding the bottleneck in Fig. 2 requires a careful consideration of the PF application. For instance, we

Table 7 Execution time and additional area of UQLE in PFs with 512 particles in the Xtensa LX2 processor.

Implementation method	Likelihood Evaluation (LE)	# of M	Average execution time (clock cycle counts)	Average speedup	Additional area (gates)
Software	ELE	–	1827.19 K	1×	0
Software with FPU	ELE	–	1734.22 K	1.05×	49 K
ASIPs	UQLE in Fixed-point	4	2.01 K	909.05×	0.57 K
		32	2.01 K	909.05×	3.75 K
		64	2.01 K	909.05×	6.60 K

know that the bottleneck of the UNG model is in the TP block. The BOT model has a complex operation, the triangle function, in the PMP block. With this approach, we can focus on optimizing the TP block in the UNG and the PMP block in BOT model. Regardless, the likelihood evaluation is still a time consuming algorithm. For applications that require very high throughput, the UQLE instruction is a promising step to improve the throughput and reduce the energy consumption. From Table 8, using the UQLE instruction results in 1.7× and 1.9× average speedup over the software implementations of the UNG and BOT models, respectively.

6 Conclusion

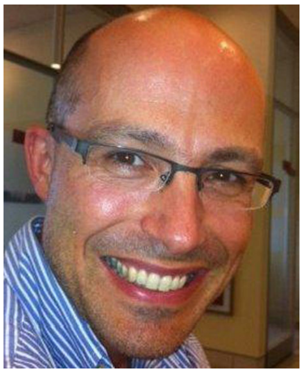
In this paper, a novel PF functional view is constructed. It focuses on distinguishing the blocks defined by the application, the algorithms or others. Under this characteristic, we demonstrate that the likelihood evaluation is a time consuming block and is not affected by the target applications. In order to speed up the execution of the likelihood evaluation, we presented an efficient uniform quantization likelihood evaluation algorithm. We then built an ASIP UQLE instruction to improve its throughput in a custom processor. Simulation results demonstrate that PFs using the proposed UQLE can achieve equal or better performance than particle filters using the exact likelihood evaluation. They also demonstrate that the proposed ASIP UQLE instruction for the BOT model can achieve an average speedup of 910× in comparison with ELE implemented in a GPP, while the processor size only increases 4.7 % for UQLE with 32 intervals.

Acknowledgements This work was supported by the GEOIDE Network Center of Excellence.

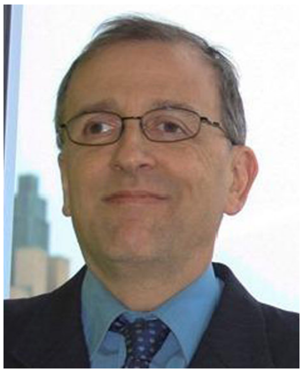
References

1. Doucet, A., Godsill, S., & Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing, 10*(3), 197–208.
2. Doucet, A., De Freitas, N., & Gordon, N. (2001). *Sequential Monte Carlo methods in practice*. New York, NY, USA: Springer.
3. Ristic, B., Arulampalam, S., & Gordon, N. (2004). *Beyond the Kalman filter: Particle filters for tracking applications*. Boston, MA, USA: Artech House.
4. Cappe, O., Godsill, S. J., & Moulines, E. (2007). An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE, 95*(5), 899–924.
5. Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). Novel-approach to nonlinear non-Gaussian Bayesian state estimation. *IEEE Proceedings-F Radar and Signal Processing, 140*(2), 107–113.
6. Gordon, N., Salmond, D., & Ewing, C. (1995). Bayesian state estimation for tracking and guidance using the bootstrap filter. *Journal of Guidance Control and Dynamics, 18*(6), 1434–1443.
7. Isard, M., & Blake, A. (1998). CONDENSATION—conditional density propagation for visual tracking. *International Journal of Computer Vision, 29*(1), 5–28.
8. Farah, R., Gan, Q., Langlois, J. M. P., Bilodeau, G. A., & Savaria, Y. (2011). A tracking algorithm suitable for embedded systems implementation. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*.
9. Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., et al. (2002). Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing, 50*(2), 425–437.
10. Gustafsson, F. (2010). Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine, 25*(7), 53–81.
11. Kalman, R. E. (1960). A new approach to linear filtering and prediction problem. *Journal of Basic Engineering Transactions, 82*, 34–45.
12. Anderson, B., & Moore, J. (1979). *Optimal filtering*. New Jersey, USA: Prentice-Hall Englewood Cliffs.
13. Culler, D. E., Gupta, A., & Singh, J. P. (1999). *Parallel computer architecture: A hardware/software approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA.
14. Bolic, M. (2004). *Architectures for efficient implementation of particle filters*. New York: State University of New York at Stony Brook.
15. Bolic, M., Djuric, P., & Hong, S. (2005). Resampling algorithms and architectures for distributed particle filters. *IEEE Transactions on Signal Processing, 53*(7), 2442–2450.
16. Hong, S., Chin, S., Djuric, P., & Bolic, M. (2006). Design and implementation of flexible resampling mechanism for high-speed parallel particle filters. *Journal of VLSI Signal Processing, 44*(1–2), 47–62.
17. Hong, S., Shi, Z., Chen, J., & Chen, K. (2010). A low-power memory-efficient resampling architecture for particle filters. *Circuits Systems and Signal Processing, 29*(1), 155–167.
18. Bolic, M., Athalye, A., Djuric, P., & Hong, S. (2004). Algorithmic modification of particle filters for hardware implementation. In *Proceedings of the European Signal Processing Conference*, Vienna, Austria. pp.1641–1644
19. Hendeby, G., Karlsson, R., & Gustafsson, F. (2010). Particle filtering: the need for speed. *Eurasip Journal on Advances in Signal Processing, 2010*, 1–9.
20. Noori, H., Mehdipour, F., Murakami, K., Inoue, K., & Zamani, M. (2008). An architecture framework for an adaptive extensible processor. *Journal of Supercomputing, 45*(3), 313–340.
21. Tensilica Inc. (2007). Xtensa LX microprocessor data book for Xtensa LX2 processor cores. Santa Clara, CA.
22. Kitagawa, G. (1996). Monte Carlo filter and smoother for Non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics, 5*(1), 1–25.
23. Tensilica Inc. (2002). Tensilica instruction extension (TIE) language user's guide. Santa Clara, CA.

Qifeng Gan received the B.Eng. in electrical engineering and automation from Shanghai University, China, in 2003 and the M.Eng. in electrical and computer engineering from Concordia University, Canada in 2008. He is currently pursuing the Ph.D. degree in computer engineering at École Polytechnique de Montréal, Canada. His current research is related to the development of efficient parallel processing algorithms for statistical signal processing and their efficient implementation on configurable processors with applications in computer vision.



J. M. Pierre Langlois was born in Sherbrooke, QC, Canada, in 1967. He received the B.Eng. degree in electrical engineering and the M.Eng. and Ph.D. degrees in computer engineering from the Royal Military College of Canada (RMC), Kingston, Ontario, Canada, in 1990, 1999, and 2002, respectively. He served as an Engineering Officer with the Canadian Navy from 1990 until 1997. He was an Assistant Professor with the Department of Electrical and Computer Engineering, RMC, from 2000 until 2005. Since 2005, he has been an Associate Professor with the Department of Computer and Software Engineering, École Polytechnique de Montréal. Dr. Langlois has authored or co-authored more than 60 journal and refereed conference papers. His current research interests include the implementation of DSP and video processing algorithms on configurable processors, with applications in computer vision, image enhancement, security and indoor navigation. Dr. Langlois is a member of the Regroupement Stratégique en Microsystèmes du Québec, the Ordre des Ingénieurs du Québec and the IEEE.



Yvon Savaria FIEEE (S' 77, M' 86, SM' 97, F'08) received the B.Ing. and M.Sc.A in electrical engineering from École Polytechnique Montreal in 1980 and 1982 respectively. He also received the Ph.D. in electrical engineering in 1985 from McGill University. Since 1985, he has been with Polytechnique Montréal, where he is currently professor and Chairman of the department of electrical engineering.

He has carried work in several areas related to microelectronic circuits and microsystems such as testing, verification, validation, clocking methods, defect and fault tolerance, effects of radiation on electronics, high-speed interconnects and circuit design techniques, CAD methods, reconfigurable computing and applications of microelectronics to telecommunications, aerospace, image processing, video processing, radar signal processing, and digital signal processing acceleration. He is currently involved in several projects that relate to aircraft embedded systems, green IT, wireless sensor network, virtual network, computational efficiency and application specific architecture design. He holds 16 patents, has published 105 journal papers and 365 conference papers, and he was the thesis advisor of 150 graduate students who completed their studies.

He was program co-chairman of ASAP'2006 and the general co-chair of ASAP'2007. He has been working as a consultant or was sponsored for carrying research by Bombardier, CNRC, Design Workshop, Dolphin, DREO, Genesis, Gennum, Hyperchip, ISR, LTRIM, Miranda, MiroTech, Nortel, Octasic, PMC-Sierra, Technocap, Thales,Tundra and VXP. He is a member of the Regroupement Stratégique en Microélectronique du Québec (RESMIQ), of the Ordre des Ingénieurs du Québec (OIQ), and was a member of CMC Microsystems board since 1999 and chairman of that board from 2008 to 2010. He was awarded in 2001 a Tier 1Canada Research Chair (http://www.chairs.gc.ca) on design and architectures of advanced microelectronic systems. He also received in 2006 a Synergy Award of the Natural Sciences and Engineering Research Council of Canada.