EURASIP Journal on Applied Signal Processing 2004:15, 2267–2277

Resampling Algorithms for Particle Filters: A Computational Complexity Perspective

Miodrag Bolić
Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794-2350, USA
Email: mbolic@ece.sunysb.edu

Petar M. Djurić
Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794-2350, USA
Email: djuric@ece.sunysb.edu

Sangjin Hong
Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794-2350, USA
Email: snjhong@ece.sunysb.edu

Received 30 April 2003; Revised 28 January 2004

Newly developed resampling algorithms for particle filters suitable for real-time implementation are described and their analysis is presented. The new algorithms reduce the complexity of both hardware and DSP realization through addressing common issues such as decreasing the number of operations and memory access. Moreover, the algorithms allow for use of higher sampling frequencies by overlapping in time the resampling step with the other particle filtering steps. Since resampling is not dependent on any particular application, the analysis is appropriate for all types of particle filters that use resampling. The performance of the algorithms is evaluated on particle filters applied to bearings-only tracking and joint detection and estimation in wireless communications. We have demonstrated that the proposed algorithms reduce the complexity without performance degradation.

Keywords and phrases: particle filters, resampling, computational complexity, sequential implementation.

1. INTRODUCTION

Particle filters (PFs) are very suitable for nonlinear and/or non-Gaussian applications. In their operation, the main principle is recursive generation of random measures, which approximate the distributions of the unknowns. The random measures are composed of particles (samples) drawn from relevant distributions and of importance weights of the particles. These random measures allow for computation of all sorts of estimates of the unknowns, including minimum mean square error (MMSE) and maximum a posteriori (MAP) estimates. As new observations become available, the particles and the weights are propagated by exploiting Bayes theorem and the concept of sequential importance sampling [1, 2].

The main goals of this paper are the development of resampling methods that allow for increased speeds of PFs, that require less memory, that achieve fixed timings regardless of the statistics of the particles, and that are computationally less complex. Development of such algorithms is extremely critical for practical implementations. The performance of the algorithms is analyzed when they are executed on a digital signal processor (DSP) and specially designed hardware. Note that resampling is the only PF step that does not depend on the application or the state-space model. Therefore, the analysis and the algorithms for resampling are general.

From an algorithmic standpoint, the main challenges include development of algorithms for resampling that are suitable for applications requiring temporal concurrency.[1] A possibility of overlapping PF operations is considered because it directly affects hardware performance, that is, it increases speed and reduces memory access. We investigate sequential resampling algorithms and analyze their computational complexity metrics including the number of operations as well as the class and type of operation by performing behavioral profiling [3]. We do not consider fixed point

[1] Temporal concurrency quantifies the expected number of operations that are simultaneously executed, that is, are overlapped in time.

precision issues, where a hardware solution of resampling suitable for fixed precision implementation has already been presented [4].

The analysis in this paper is related to the sample importance resampling (SIR) type of PFs. However, the analysis can be easily extended to any PF that performs resampling, for instance, the auxiliary SIR (ASIR) filter. First, in Section 2 we provide a brief review of the resampling operation. We then consider random and deterministic resampling algorithms as well as their combinations. The main feature of the random resampling algorithm, referred to as residual-systematic resampling (RSR) and described in Section 3, is to perform resampling in fixed time that does not depend on the number of particles at the output of the resampling procedure. The deterministic algorithms, discussed in Section 4, are threshold-based algorithms, where particles with moderate weights are not resampled. Thereby significant savings can be achieved in computations and in the number of times the memories are accessed. We show two characteristic types of deterministic algorithms: a low-complexity algorithm and an algorithm that allows for overlapping of the resampling operation with the particle generation and weight computation. The performance and complexity analysis are presented in Section 5 and the summary of our contributions is outlined in Section 6.

2. OVERVIEW OF RESAMPLING IN PFs

PFs are used for tracking states of dynamic state-space models described by the set of equations

$$\begin{aligned} \mathbf{x}_n &= f(\mathbf{x}_{n-1}) + \mathbf{u}_n, \\ \mathbf{y}_n &= g(\mathbf{x}_n) + \mathbf{v}_n, \end{aligned} \tag{1}$$

where $\mathbf{x}_n$ is an evolving state vector of interest, $\mathbf{y}_n$ is a vector of observations, $\mathbf{u}_n$ and $\mathbf{v}_n$ are independent noise vectors with known distributions, and $f(\cdot)$ and $g(\cdot)$ are known functions. The most common objective is to estimate $\mathbf{x}_n$ as it evolves in time.

PFs accomplish tracking of $\mathbf{x}_n$ by updating a random measure $\{\mathbf{x}_{1:n}^{(m)}, w_n^{(m)}\}_{m=1}^{M}$,[2] which is composed of M particles $\mathbf{x}_n^{(m)}$ and their weights $w_n^{(m)}$ defined at time instant n, recursively in time [5, 6, 7]. The random measure approximates the posterior density of the unknown trajectory $\mathbf{x}_{1:n}$, $p(\mathbf{x}_{1:n}|\mathbf{y}_{1:n})$, where $\mathbf{y}_{1:n}$ is the set of observations.

In the implementation of PFs, there are three important operations: particle generation, weight computation, and resampling. Resampling is a critical operation in particle filtering because with time, a small number of weights dominate the remaining weights, thereby leading to poor approximation of the posterior density and consequently to inferior estimates. With resampling, the particles with large weights are replicated and the ones with negligible weights are removed. After resampling, the future particles are more concentrated in domains of higher posterior probability, which entails improved estimates.

The PF operations are performed according to

(1) generation of particles (samples) $\mathbf{x}_n^{(m)} \sim \pi(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i_{n-1}^m)}, \mathbf{y}_{1:n})$, where $\pi(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i_{n-1}^m)}, \mathbf{y}_{1:n})$ 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

$$w_n^{*(m)} = \frac{w_{n-1}^{(i_{n-1}^m)}}{a_{n-1}^{(i_{n-1}^m)}} \frac{p(\mathbf{y}_n|\mathbf{x}_n^{(m)})\, p(\mathbf{x}_n^{(m)}|\mathbf{x}_{n-1}^{(i_{n-1}^m)})}{\pi(\mathbf{x}_n^{(m)}|\mathbf{x}_{n-1}^{(i_{n-1}^m)}, \mathbf{y}_{1:n})} \tag{2}$$

followed by normalization $w_n^{(m)} = w_n^{*(m)} / \sum_{j=1}^{M} w_n^{*(j)}$;

(3) resampling $i_n^{(m)} \sim a_n^{(m)}$, where $a_n^{(m)}$ is a suitable resampling function whose support is defined by the particle $x_n^{(m)}$ [8].

The above representation of the PF algorithm provides a certain level of generality. For example, the SIR filter with a stratified resampling is implemented by choosing $a_n^{(m)} = w_n^{(m)}$ for $m = 1, \ldots, M$. When $a_n^{(m)} = 1/M$, there is no resampling and $i_n^{(m)} = m$. The ASIR filter can be implemented by setting $a_n^{(m)} = w_n^{(m)} p(\mathbf{y}_{n+1}|\boldsymbol{\mu}_{n+1}^{(m)})$ and $\pi(\mathbf{x}_n) = p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(m)})$, where $\boldsymbol{\mu}_n^{(m)}$ is the mean, the mode, or some other likely value associated with the density $p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(m)})$.

3. RESIDUAL SYSTEMATIC RESAMPLING ALGORITHM

In this section, we consider stratified random resampling algorithms, where $a_n^{(m)} = w_n^{(m)}$ [9, 10, 11]. Standard algorithms used for random resampling are different variants of stratified sampling such as residual resampling (RR) [12], branching corrections, [13] and systematic resampling (SR) [6]. SR is the most commonly used since it is the fastest resampling algorithm for computer simulations.

We propose a new resampling algorithm which is based on stratified resampling, and we refer to it as RSR [14]. Similar to RR, RSR calculates the number of times each particle is replicated except that it avoids the second iteration of RR when residual particles need to be resampled. Recall that in RR, the number of replications of a specific particle is determined in the first loop by truncating the product of the number of particles and the particle weight. In RSR instead, the updated uniform random number is formed in a different fashion, which allows for only one iteration loop and processing time that is independent of the distribution of the weights at the input. The RSR algorithm for N input and M output (resampled) particles is summarized by Algorithm 1.

Figure 1 graphically illustrates the SR and RSR methods for the case of $N = M = 5$ particles with weights given in Table 1. SR calculates the cumulative sum of the weights $C^{(m)} = \Sigma_{i=1}^{m} w_n^{(i)}$ and compares $C^{(m)}$ with the updated uniform number $U^{(m)}$ for $m = 1, \ldots, N$. The uniform number

[2] The notation $\mathbf{x}_{1:n}$ signifies $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.

Purpose: generation of an array of indexes $\{i\}_1^N$ at time instant n, $n > 0$.
Input: an array of weights $\{w_n\}_1^N$, input and output number of particles, N and M, respectively.
Method:
$(i) = \mathrm{RSR}(N, M, w)$
Generate a random number $\Delta U^{(0)} \sim \mathcal{U}[0, 1/M]$
for $m = 1$–N
$i^{(m)} = \lfloor (w_n^{(m)} - \Delta U^{(m-1)}) \cdot M \rfloor + 1$
$\Delta U^{(m)} = \Delta U^{(m-1)} + i^{(m)}/M - w_n^{(m)}$
end

Algorithm 1: Residual systematic resampling algorithm.

$U^{(0)}$ is generated by drawing from the uniform distribution $\mathcal{U}[0, 1/M]$ and updated by $U^{(m)} = U^{(m-1)} + 1/M$. The number of replications for particle m is determined as the number of times the updated uniform number is in the range $[C^{(m-1)}, C^{(m)})$. For particle 1, $U^{(0)}$ and $U^{(1)}$ belong to the range $[0, C^{(1)})$, so that this particle is replicated twice, which is shown with two arrows that correspond to particle 1. Particles 2 and 3 are replicated once. Particle 4 is discarded ($i^{(4)} = 0$) because no $U^{(m)}$ for $m = 1, \ldots, N$ appears in the range $[C^{(3)}, C^{(4)})$.

The RSR algorithm draws the uniform random number $U^{(0)} = \Delta U^{(0)}$ in the same way but updates it by $\Delta U^{(m)} = \Delta U^{(m-1)} + i^{(m)}/M - w_n^{(m)}$. In the figure, we display both $U^{(m)} = \Delta U^{(m-1)} + i^{(m)}/M$ and $\Delta U^{(m)} = U^{(m)} - w_n^{(m)}$. Here, the uniform number is updated with reference to the *origin of the currently considered weight*, while in SR, it is propagated with reference to the *origin of the coordinate system*. The difference $\Delta U^{(m)}$ between the updated uniform number and the current weight is propagated. Figure 1 shows that $i^{(1)} = 2$ and that $\Delta U^{(1)}$ is calculated and then used as the initial uniform random number for particle 2. Particle 4 is discarded because $\Delta U^{(3)} = U^{(4)} > w^{(4)}$, so that $\lfloor (w_n^{(4)} - \Delta U^{(3)}) \cdot M \rfloor = -1$ and $i^{(4)} = 0$. If we compare $\Delta U^{(1)}$ with the relative position of the $U^{(2)}$ and $C^{(1)}$ in SR, $\Delta U^{(2)}$ in RSR with the relative position of $U^{(3)}$ and $C^{(2)}$ in SR, and so on, we see that they are equal. Therefore, SR and RSR produce identical resampling result.

3.1. Particle allocation and memory usage

We call particle allocation the way in which particles are placed in their new memory locations as a result of resampling. With proper allocation, we want to reduce the number of memory accesses and the size of state memory. The allocation is performed through index addressing, and its execution can be overlapped in time with the particle generation step. In Figure 2, three different outputs of resampling for the input weights from Figure 1 are considered. In Figure 2a, the indexes represent positions of the replicated particles. For example, $i^{(2)} = 1$ means that particle 1 replaces particle 2. Particle allocation is easily overlapped with particle generation using $\tilde{x}^{(m)} = x^{(i^{(m)})}$ for $m = 1, \ldots, M$, where $\{\tilde{x}^{(m)}\}_{m=1}^M$ is the set of resampled particles. The randomness of the resampling output makes it difficult to realize in-place storage so that additional temporary memory for storing resampled particles $\tilde{x}^{(m)}$ is necessary. In Figure 2a, particle 1 is replicated twice and occupies the locations of particles 1 and 2. Particle 2 is replicated once and must be stored in the memory of $\tilde{x}^{(m)}$ or it would be rewritten. We refer to this method as *particle allocation with index addressing*.

In Figure 2b, the indexes represent the number of times each particle is replicated. For example, $i^{(1)} = 2$ means that particle 1 is replicated twice. We refer to this method as *particle allocation with replication factors*. This method still requires additional memory for particles and memory for storing indexes.

The additional memory for storing the particles $\tilde{x}^{(m)}$ is not necessary if the particles are replicated to the positions of the discarded particles. We call this method *particle allocation with arranged indexes* of positions and replication factors (Figure 2c). Here, the addresses of both replicated particles and discarded particles as well as the number of times they are replicated (replication factor) are stored. The indexes are arranged in a way so that the replicated particles are placed in the upper and the discarded particles in the lower part of the index memory. In Figure 2c, the replicated particles take the addresses $1 - 4$ and the discarded particle is on the address 5. When one knows in advance the addresses of the discarded particles, there is no need for additional memory for storing the resampled particles $\tilde{x}^{(m)}$ because the new particles are placed on the addresses occupied by the particles that are discarded. It is useful for PFs applied to multidimensional models since it avoids need for excessive memory for storing temporary particles.

For the RSR method, it is natural to use particle allocation with replication factor and arranged indexes because the RSR produces replication factors. In the particle generation step, the *for* loop with the number of iterations that corresponds to the replication factors is used for each replicated particle. The difference between the SR and the RSR methods is in the way the inner loop in the resampling step for SR and particle generation step for RSR is performed. Since the number of replicated particles is random, the *while* loop in SR has an unspecified number of operations. To allow for an unspecified number of iterations, complicated control structures in hardware are needed [15]. The main advantage of our approach is that the *while* loop of SR is replaced with a *for* loop with known number of iterations.

4. DETERMINISTIC RESAMPLING

4.1. Overview

In the literature, threshold-based resampling algorithms are based on the combination of RR and rejection control and they result in nondeterministic timing and increased complexity [8, 16]. Here, we develop threshold-based algorithms whose purpose is to reduce complexity and processing time. We refer to these methods as partial resampling (PR) because only a part of the particles is resampled.

In PR, the particles are grouped in two separate classes: one composed of particles with moderate weights and another with dominating and negligible weights. The particles

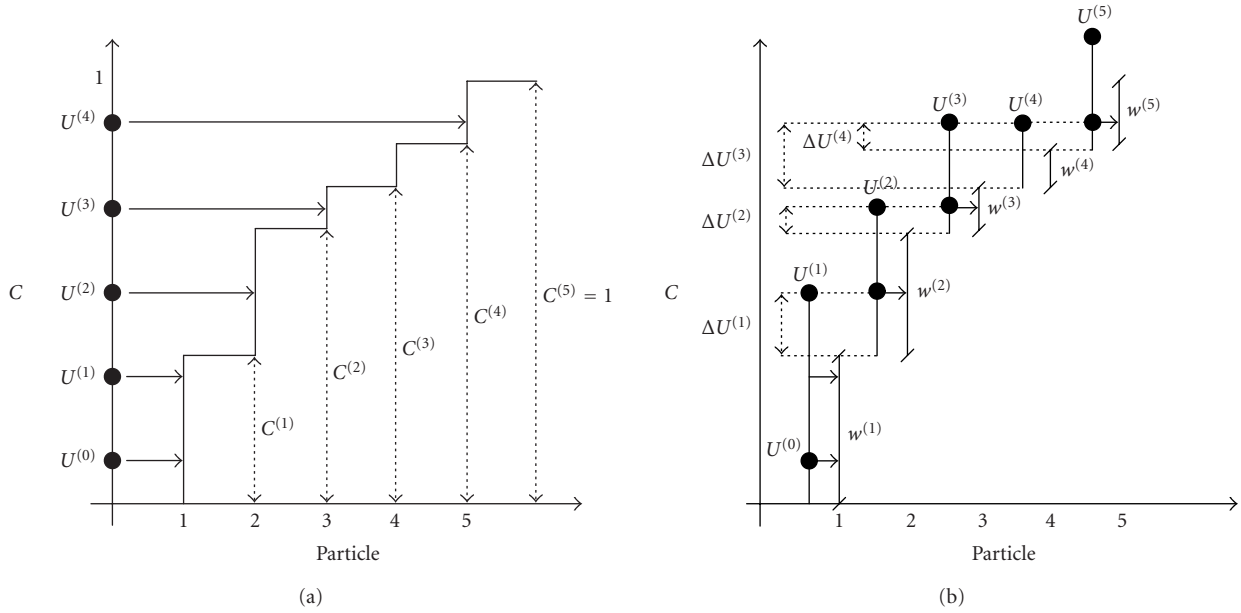


Figure 1: (a) Systematic and (b) residual systematic resampling for an example with $M = 5$ particles.

Table 1: Weights of particles.

m	$w^{(m)}$	$i^{(m)}$
1	7/20	2
2	6/20	1
3	2/20	1
4	2/20	0
5	3/20	1

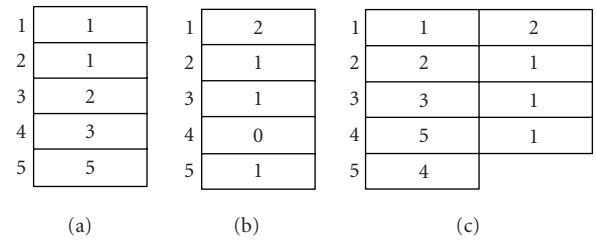


Figure 2: Types of memory usages: (a) indexes are positions of the replicated particles, (b) indexes are replication factors, and (c) indexes are arranged as positions and replication factors.

with moderate weights are not resampled, whereas the negligible and dominating particles are resampled. It is clear that on average, resampling would be performed much faster because the particles with moderate weights are not resampled. We propose several PR algorithms which differ in the resampling function.

4.2. Partial resampling: suboptimal algorithms

PR could be seen as a way of a partial correction of the variance of the weights at each time instant. PR methods consist of two steps: one in which the particles are classified as moderate, negligible, or dominating and the other in which one determines the number of times each particle is replicated. In the first step of PR, the weight of each particle is compared with a high and a low threshold, T_h and T_l, respectively, where $T_h > 1/M$ and $0 < T_l < T_h$. Let the number of particles with weights greater than T_h and less than T_l be denoted by N_h and N_l, respectively. A sum of the weights of resampled particles is computed as a sum of dominating $W_h = \sum_{m=1}^{N_h} w_n^{(m)}$ for $w_n^{(m)} > T_h$ and negligible weights $W_l = \sum_{m=1}^{N_l} w_n^{(m)}$ for $w_n^{(m)} < T_l$. We define three different types of resampling with distinct resampling functions $a_n^{(m)}$.

The resampling function of the first PR algorithm (PR1) is shown in Figure 3a and it corresponds to the stratified resampling case. The number of particles at the input and at the output of the resampling procedure is the same and equal to $N_h + N_l$. The resampling function is given by

$$a_n^{(m)} = \begin{cases} w_n^{(m)} & \text{for } w_n^{(m)} > T_h \text{ or } w_n^{(m)} < T_l, \\ \dfrac{1 - W_h - W_l}{M - N_h - N_l} & \text{otherwise.} \end{cases} \tag{3}$$

The second step can be performed using any resampling algorithm. For example, the RSR algorithm can be called using $(i) = \mathrm{RSR}(N_h+N_l, N_h+N_l, w_n^{(m)}/(W_h+W_l))$, where the RSR is performed on the $N_h + N_l$ particles with negligible and dominating weights. The weights have to be normalized before they are processed by the RSR method.

The second PR algorithm (PR2) is shown in Figure 3b. The assumption that is made here is that most of the negligible particles will be discarded after resampling, and

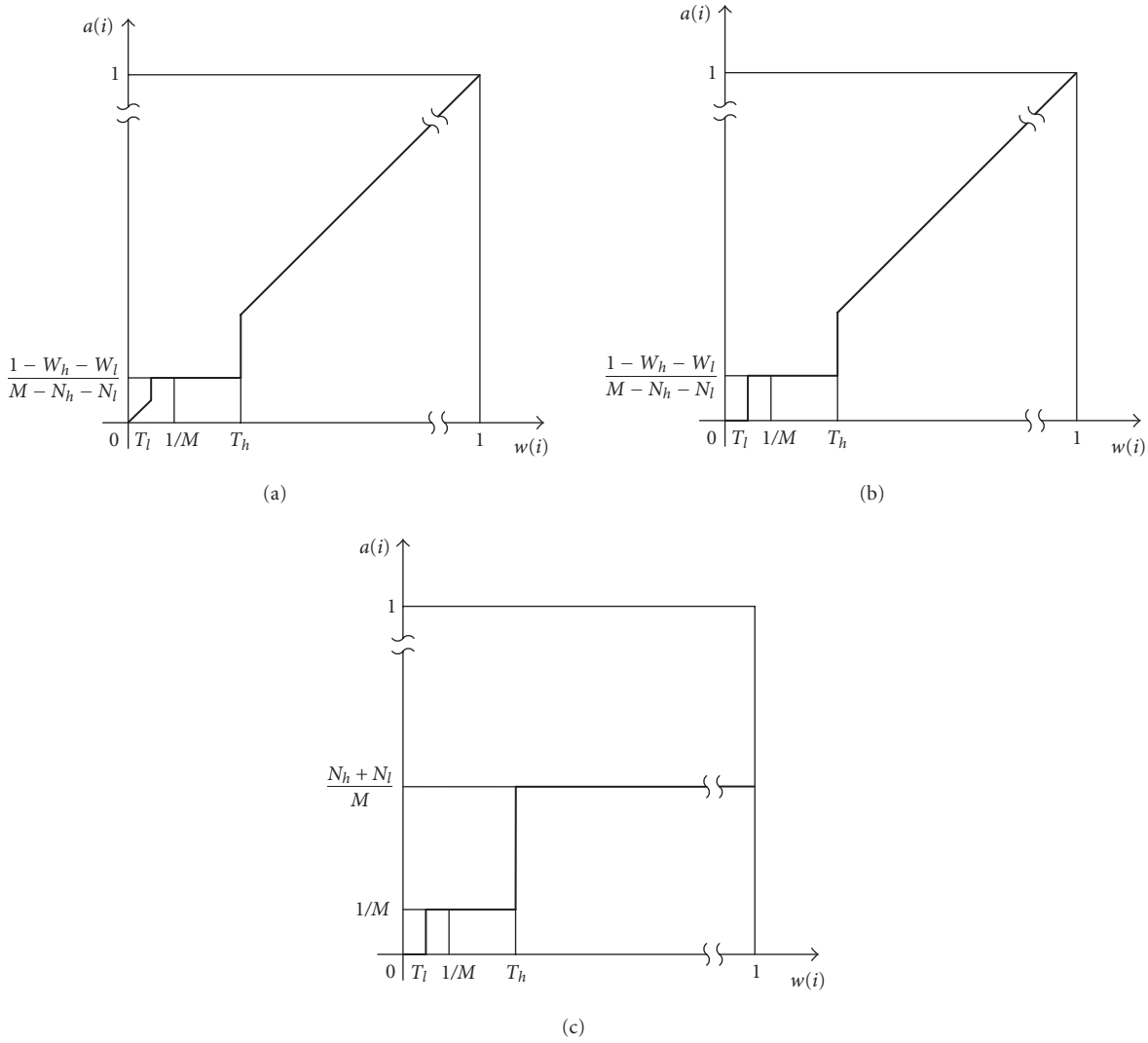


Figure 3: Resampling functions for the PR algorithms (a) PR1, (b) PR2, and (c) PR3.

consequently, particles with negligible weights are not used in the resampling procedure. Particles with dominating weights replace those with negligible weights with certainty. The resampling function is given as

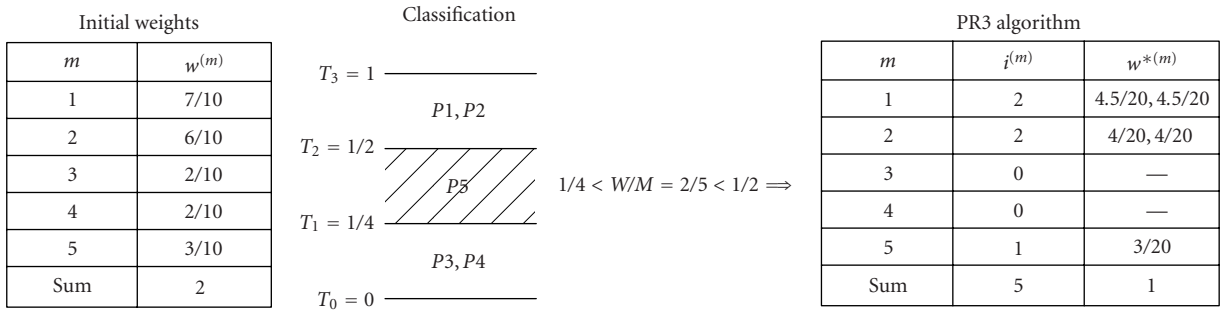
$$a_n^{(m)} = \begin{cases} w_n^{(m)} + \dfrac{W_l}{N_h} & \text{for } w_n^{(m)} > T_h, \\ \dfrac{1 - W_h - W_l}{M - N_h - N_l} & \text{for } T_l < w_n^{(m)} < T_h, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The number of times each particle is replicated can be found using $(i) = \mathrm{RSR}(N_h, N_h + N_l, (w_n^{(m)} + W_l/N_h)/(W_h + W_l))$, where the weights satisfy the condition $w_n^{(m)} > T_h$. There are only N_h input particles and $N_h + N_l$ particles are produced at the output.

The third PR algorithm (PR3) is shown in Figure 3c. The weights of all the particles above the threshold T_h are scaled with the same number. So, PR3 is a deterministic algorithm whose resampling function is given as

$$a_n^{(m)} = \begin{cases} \dfrac{N_h + N_l}{M} & \text{for } w_n^{(m)} > T_h, \\ \dfrac{1}{M} & \text{for } T_l < w_n^{(m)} < T_h, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The number of replications of each dominating particle may be less by one particle than necessary because of the rounding operation. One way of resolving this problem is to assign that the first $N_t = N_l - \lfloor N_l/N_h \rfloor N_h$ dominating particles are replicated $r = \lfloor N_l/N_h \rfloor + 2$ times, while the rest of $N_h - N_t$ dominating particles are replicated $r = \lfloor N_l/N_h \rfloor + 1$ times. The weights are calculated as $w^{*(m)} = w^{(m)}$, where m represents positions of particles with moderate weights, and as $w^{*(l)} = w^{(m)}/r + W_l/(N_h + N_l)$, where m are positions of particles with dominating weights and l of particles with both dominating and negligible weights.



m	$w^{(m)}$
1	7/10
2	6/10
3	2/10
4	2/10
5	3/10
Sum	2

m	$i^{(m)}$	$w^{*(m)}$
1	2	4.5/20, 4.5/20
2	2	4/20, 4/20
3	0	—
4	0	—
5	1	3/20
Sum	5	1

Figure 4: OPR method combined with the PR3 method used for final computation of weights and replication factors.

Another way of performing PR is to use a set of thresholds. The idea is to perform initial classification of the particles while the weights are computed and then to carry out the actual resampling together with the particle generation step. So, the resampling consists of two steps as in the PR2 algorithm, where classification of the particles is overlapped with the weight computation. We refer to this method as overlapped PR (OPR).

A problem with the classification of the particles is the necessity of knowing the overall sum of nonnormalized weights in advance. The problem can be resolved as follows. The particles are partitioned according to their weights. The thresholds for group i are defined as T_{i-1}, T_i for $i = 1, \ldots, K$, where K is the number of groups, $T_{i-1} < T_i$ and $T_0 = 0$. The selection of thresholds is problem dependent. The thresholds that define the moderate group of particles satisfy $T_{k-1} < W/M < T_k$. The particles that have weights greater than T_k are dominant particles, and the ones with weights less than T_{k-1}, negligible particles.

In Figure 4, we provide a simple example of how this works. There are four thresholds (T_0 to T_3) and nonnormalized particles are compared with the thresholds and properly grouped. After obtaining the sum of weights W, the second group for which $T_1 < W/M < T_2$, is the group of particles with moderate weights. The first group contains particles with negligible weights, and the third group is composed of particles with dominating weights. An additional loop is necessary to determine the number of times each of the dominating particles is replicated. However, the complexity of this loop is of order $O(K)$, which is several orders of magnitude lower than the complexity of the second step in the PR1 algorithm ($O(M)$). Because the weights are classified, it is possible to apply similar logic for the second resampling step as in the PR2 and PR3 algorithms. In the figure, the particles P1 and P2 are replicated twice and their weights are calculated using the formulae for weights for the PR3 method.

4.3. Discussion

In the PR1, PR2, and PR3 algorithms, the first step requires a loop of M iterations for the worst case (of number of computations) with two comparisons per each iteration (classification in three groups). Resampling in the PR1 algorithm is performed on $N_l + N_h$ particles. The worst case for the PR1 algorithm occurs when $N_l + N_h = M$, which means that all the particles must be resampled, thereby implying that there cannot be improvements from an implementation standpoint. The main purpose of the PR2 algorithm is to improve the worst-case timing of the PR1 algorithm. Here, only N_h dominating particles are resampled. So, the input number of particles in the resampling procedure is N_h, while the output number of particles is $N_h + N_l$. If the RSR algorithm is used for resampling, then the complexity of the second step is $O(N_h)$.

PR1 and PR2 contain two loops and their timings depend on the weight statistics. As such, they do not have advantages for real-time implementation in comparison with RSR, which has only one loop of M iterations and whose processing time does not depend on the weight statistics. In the PR3 algorithm, there is no stratified resampling. The number of times each dominating particle is replicated is calculated after the first step and it depends on the current distribution of particle weights and of the thresholds. This number is calculated in $O(1)$ time, which means that there is no need for another loop in the second step. Thus, PR3 has simpler operations than the RSR algorithm.

The PR algorithms have the following advantages from the perspective of hardware implementation: (1) the resampling is performed faster on average because it is done on a much smaller number of particles; (2) there is a possibility of overlapping the resampling with the particle generation and weight computation; and (3) if the resampling is used in a parallel implementation [17], the number of exchanged particles among the processing elements is smaller because there are less particles to be replicated and replaced. There are also problems with the three algorithms. When $N_l = 0$ and $N_h = 0$, resampling is not necessary. However, when $N_l = 0$ or $N_h = 0$ but not at same time, the PR algorithms would not perform resampling even though it could be useful.

Application of the OPR algorithm requires a method for fast classification. For hardware and DSP implementation, it is suitable to define thresholds that are a power of two. So, we take that $T_i = 1/2^{K-i}$ for $i = 1, \ldots, K$ and $T_0 = 0$. The group is determined by the position of the most significant "one" in the fixed point representation of weights. Memory allocation for the groups could be static or dynamic. Static allocation requires K memory banks, where the size of each bank is equal to the number of particles because all the particles could be located in one of the groups. Dynamic allocation is

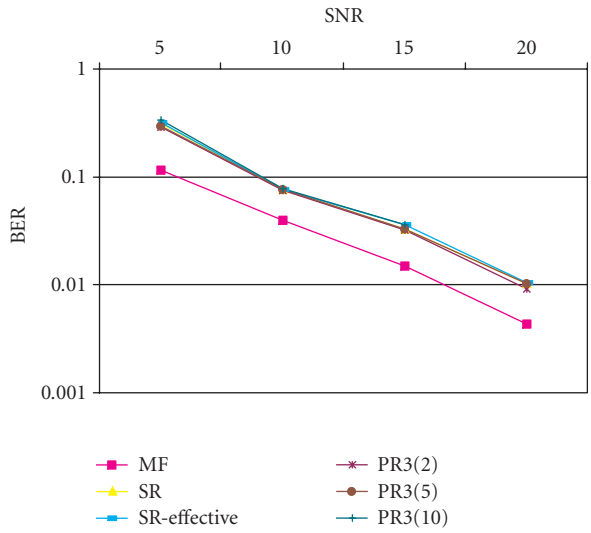


FIGURE 5: Performance of the PR3 algorithm for different threshold values applied to joint detection and estimation.

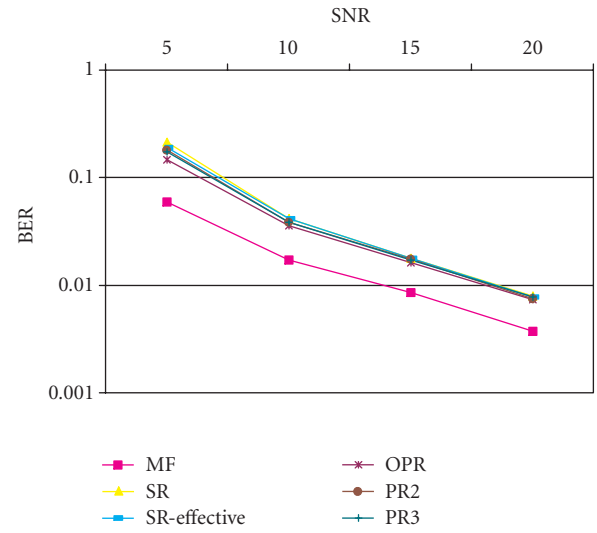


FIGURE 6: Comparison of the PR2, PR3, and OPR algorithms with SR applied to the joint detection and estimation problem.

more efficient and it could be implemented using ways similar to the linked lists, where the element in a group contains two fields: the field with the address of the particle and the field that points out to the next element on the list. Thus, dynamic allocation requires memory with capacity of $2M$ words. As expected, overlapping increases the resources.

5. PARTICLE FILTERING PERFORMANCE AND COMPLEXITY

5.1. Performance analysis

The proposed resampling algorithms are applied and their performance is evaluated for the joint detection and estimation problem in communication [18, 19] and for the bearings-only tracking problem [7].

5.1.1. Joint detection and estimation

The experiment considered a Rayleigh fading channel with additive Gaussian noise with a differentially encoded BPSK modulation scheme. The detector was implemented for a channel with normalized Doppler spreads given by $B_d = 0.01$, which corresponds to fast fading. An AR(3) process was used to model the channel. The AR coefficients were obtained from the method suggested in [20]. The proposed detectors were compared with the *clairvoyant* detector, which performs matched filtering and detection assuming that the channel is known exactly by the receiver. The number of particles was $N = 1000$.

In Figure 5, the bit error rate (BER) versus signal-to-noise ratio (SNR) is depicted for the PR3 algorithm with different sets of thresholds, that is, $T_h = \{2M, 5M, 10M\}$ and $T_l = \{1/(2M), 1/(5M), 1/(10M)\}$. In the figure, the PR3 algorithm with the thresholds $2M$ and $1/2M$ is denoted as PR3(2), the one with thresholds $5M$ and $1/5M$ as PR3(5), and so on. The BERs for the matched filter (MF) and for the case when the SR is performed are shown as well. It is observed that the BER is similar for all types of resampling. However, the best results are obtained when the thresholds $2M$ and $1/2M$ were used. Here, the effective number of particles that is used is the largest in comparison with the PR3 algorithm with greater T_h and smaller T_l. This is a logical result because according to PR3, all the particles are concentrated in the narrower area between the two thresholds producing in this way a larger effective sample size. PR3 with thresholds $2M$ and $1/2M$ slightly outperforms the SR algorithm which is a bit surprising. The reason for this could be that the particles with moderate weights are not unnecessarily resampled in the PR3 algorithm. The same result is obtained even with different values of Doppler spread.

In Figure 6, BER versus SNR is shown for different resampling algorithms: PR2, PR3, OPR, and SR. The thresholds that are used for the PR2 and PR3 are $2M$ and $1/2M$. The OPR uses $K = 24$ groups and thresholds which are power of two. Again, all the results are comparable. The OPR and PR2 algorithms slightly outperform the other algorithms.

5.1.2. Bearings-only tracking

We tested the performance of PFs by applying the resampling algorithms to bearings-only tracking [7] with different initial conditions. In the experiment, PR2 and PR3 are used with two sets of threshold values, that is, $T_h = \{2M, 10M\}$ and $T_l = \{1/(2M), 1/(10M)\}$. In Figure 7, we show the number of times when the track is lost versus number of particles, for two different pairs of thresholds. We consider that the track is lost if all the particles have zero weights. In the figure, the PR3 algorithm with thresholds $2M$ and $1/2M$ is denoted as PR3(2) and the one $10M$ and $1/10M$ as PR3(10). The used algorithms are SR, SR performed after every 5th observation, PR2, and PR3. The resampling algorithms show again similar performances. The best results for PR2 and PR3 are obtained when the thresholds $10M$ and $1/10M$ are used.

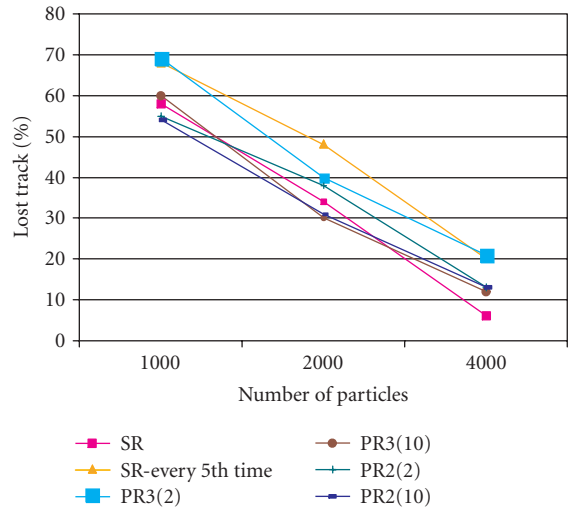


Figure 7: Number of times when the track is lost for the PR2, PR3, and SR applied to the bearings-only tracking problem.

5.2. Complexity analysis

The complexity of the proposed resampling algorithms is evaluated. We consider both computation complexity as well as memory requirements. We also present benefits of the proposed algorithms when concurrency in hardware is exploited.

5.2.1. Computational complexity

In Table 2, we provide a comparison of the different resampling algorithms. The results for RR are obtained for the worst-case scenario. The complexity of the RR, RSR, and PR algorithms is of $O(N)$, and the complexity of the SR algorithm is of $O(\max(N, M))$, where N and M are the input and output numbers of particles of the resampling procedure.

When the number of particles at the input of the resampling algorithm is equal to the number of particles at the output, the RR algorithm is by far the most complex. While the number of additions for the SR and RSR algorithms are the same, the RSR algorithm performs M multiplications. Since multiplication is more complex than addition, we can view that the SR is a less complex algorithm. However, when N is a power of two such that the multiplications by N are avoided, the RSR algorithm is the least complex.

The resampling algorithms SR, RSR, and PR3 were implemented on the Texas Instruments (TI) floating-point DSP TMS320C67xx. Several steps of profiling brought about five-fold speed-up when the number of resampled particles was 1000. The particle allocation step was not considered. The number of clock cycles per particle was around 18 for RSR and 4.1 for PR3. The SR algorithm does not have fixed timing. The mean duration was 24.125 cycles per particle with standard deviation of 5.17. On the processor TMS320C6711C whose cycle time is 5 nanoseconds, the processing of RSR with 1000 particles took 90 microseconds.

Table 2: Comparison of the number of operations for different resampling algorithms.

Operation	SR	RR	RSR	PR3
Multiplications	0	N	N	0
Additions	$2M + N$	$6N$	$3N$	$2N$
Comparisons	$N + M$	$3N$	0	$2N$

5.2.2. Memory requirements

In our analysis, we considered the memory requirement not only for resampling, but also for the complete PF. The memory size of the weights and the memory access during weight computation do not depend on the resampling algorithm. We consider particle allocation without indexes and with index addressing for the SR algorithm, and with arranged indexing for RSR, PR2, PR3, and OPR. For both particle allocation methods, the SR algorithm has to use two memories for storing particles. In Table 3, we can see the memory capacity for the RSR, PR2, and PR3 algorithms. The difference among these methods is only in the size of the index memory. For the RSR algorithm which uses particle allocation with arranged indexes, the index memory has a size of $2M$, where M words are used for storing the addresses of the particles that are replicated or discarded. The other M words represent the replication factors.

The number of resampled particles for the worst case of the PR2 algorithm corresponds to the number of particles in the RSR algorithm. Therefore, their index memories are of the same size. From an implementation standpoint, the most promising algorithm is the PR3 algorithm. It is the simplest one and it requires the smallest size of memory. The replication factor of the dominating particles is the same and of the moderate particles is one. So, the size of the index memory of PR3 is M, and it requires only one additional bit to represent whether a particle is dominant or moderate.

The OPR algorithm needs the largest index memory. When all the PF steps are overlapped, it requires a different access pattern than the other deterministic algorithms. Due to possible overwriting of indexes that are formed during the weight computation step with the ones that are read during particle generation, it is necessary to use two index-memory banks. Furthermore, particle generation and weight computation should access these memories alternately. Writing to the first memory is performed in the resampling step in one time instance whereas in the next one, the same memory is used by particle generation for reading. The second memory bank is used alternately. If we compare the memory requirements of the OPR algorithm with that of the PR3 algorithm, it is clear that OPR requires four times more memory for storing indexes for resampling.

5.2.3. PF speed improvements

The PF sampling frequency can be increased in hardware by exploiting temporal concurrency. Since there are no data dependencies among the particles in the particle generation and weight computation, the operations of these two steps

Table 3: Memory capacity for different resampling algorithms.

	SR without indexes	SR with indexes	RSR	PR2	PR3	OPR
States	$2N_sM$	$2N_sM$	N_sM	N_sM	N_sM	N_sM
Weights	M	M	M	M	M	M
Indexes	0	M	$2M$	$2M$	M	$4M$

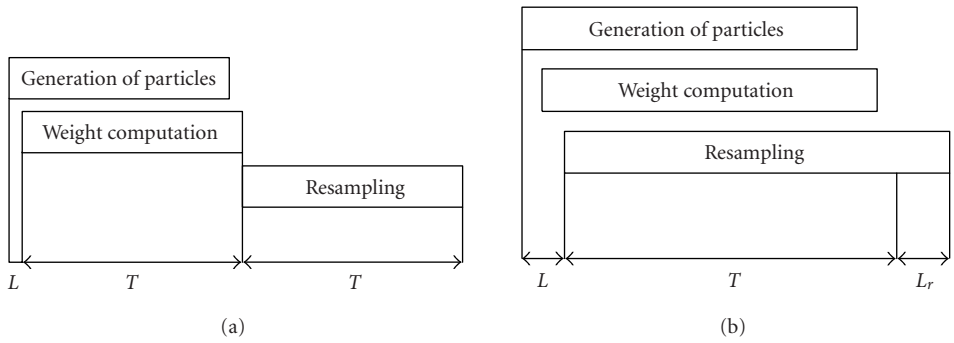


Figure 8: The timing of the PF with the (a) RSR or PR methods and (b) with the OPR method.

can be overlapped. Furthermore, the number of memory accesses is reduced because during weight computation, the values of the states do not need to be read from the memory since they are already in the registers.

The normalization step requires the use of an additional loop of M iterations as well as M divisions per observation. It has been noted that the normalization represents an unnecessary step which can be merged with the resampling and/or the computation of the importance weights. Avoidance of normalization requires additional changes which depend on whether resampling is carried out at each time instant and on the type of resampling. For PFs which perform SR or RSR at each time instant, the uniform random number in the resampling algorithm should be drawn from $[0, W_M/M)$ and updated with W_M/M, where W_M is the sum of the weights. Normalization in the PR methods could be avoided by including information about the sum W_M in the thresholds by using $T_{hn} = T_h W_M$ and $T_{ln} = T_l W_M$. With this approach, dynamic range problems for fixed precision arithmetics that appear usually with division are reduced. The computational burden is decreased as well because the number of divisions is reduced from M to 1.

The timing operations for a hardware implementation, where all the blocks are fine-grain pipelined are shown in Figure 8a. Here, the particle generation and weight calculation operations are overlapped in time and normalization is avoided. The symbol L is the constant hardware latency defined by the depth of pipelining in the particle generation and weight computation, T_{clk} is the clock period, M is the number of particles, and T is the minimum processing time of any of the basic PF operations. The SR is not suitable for hardware implementations, where fixed and minimal timings are required, because its processing time depends on the weight distribution and it is longer than MT_{clk}. So, in order to have resampling operation performed in M clock cycles, RSR or PR3 algorithms with particle allocation with arranged indexes must be used. The minimum PF sampling period that can be achieved is $(2MT_{\text{clk}} + L)$.

OPR in combination with the PR3 algorithm allows for higher sampling frequencies. In the OPR, the classification of the particles is overlapped with the weight calculation as shown in Figure 8b. The symbol L_r is the constant latency of the part of the OPR algorithm that determines which group contains moderate, and which contains negligible and dominating particles. The latency L_r is proportional to the number of OPR groups. The speed of the PF can almost be increased twice if we consider pipelined hardware implementation. In Figure 8b, it is obvious that the PF processing time is reduced to $(MT_{clk} + L + L_r)$.

5.3. Final remarks

We summarize the impact of the proposed resampling algorithms on the PF speed and memory requirements.

(1) The RSR is an improved RR algorithm with higher speed and fixed processing time. As such, besides for hardware implementations, it is a better algorithm for resampling that is executed on standard computers.

(2) Memory requirements are reduced. The number of memory access and the size of the memory are reduced when RSR or any of PR algorithms are used for multidimensional state-space models. These methods can be appropriate for both hardware and DSP applications, where the available memory is limited. When the state-space model is one dimensional, then there is no purpose of adding an index memory and introducing a more complex control. In this case, the SR algorithm is recommended.

(3) In hardware implementation and with the use of temporal concurrency, the PF sampling frequency can be considerably improved. The best results are achieved for the OPR algorithm at the expense of hardware resources.

(4) The average amount of operations is reduced. This is true for PR1, PR2, and PR3 since they perform resampling on a smaller number of particles. This is desirable in PC simulations and some DSP applications.

6. CONCLUSION

Resampling is a critical step in the hardware implementation of PFs. We have identified design issues of resampling algorithms related to execution time and storage requirement. We have proposed new resampling algorithms whose processing time is not random and that are more suitable for hardware implementation. The new resampling algorithms reduce the number of operations and memory access or allow for overlapping the resampling step with weight computation and particle generation. While these algorithms minimize performance degradation, their complexity is reduced remarkably. We have also provided performance analysis of PFs that use our resampling algorithms when applied to joint detection and estimation in wireless communications and bearings-only tracking. Even though the algorithms are developed with the aim of improving the hardware implementation, these algorithms should also be considered as resampling methods in simulations on standard computers.

ACKNOWLEDGMENT

This work has been supported under the NSF Awards CCR-0082607 and CCR-0220011.

REFERENCES

[1] J. M. Bernardo and A. F. M. Smith, *Bayesian Theory*, John Wiley & Sons, New York, NY, USA, 1994.

[2] J. Geweke, "Antithetic acceleration of Monte Carlo integration in Bayesian inference," *Journal of Econometrics*, vol. 38, no. 1-2, pp. 73–89, 1988.

[3] A. Gerstlauer, R. Domer, J. Peng, and D. Gajski, *System Design: A Practical Guide with SpecC*, Kluwer Academic Publishers, Boston, Mass, USA, 2001.

[4] S. Hong, M. Bolić, and P. M. Djurić, "An efficient fixed-point implementation of residual resampling scheme for high-speed particle filters," *IEEE Signal Processing Letters*, vol. 11, no. 5, pp. 482–485, 2004.

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

[6] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*, Springer, New York, NY, USA, 2001.

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

[8] J. S. Liu, R. Chen, and T. Logvinenko, "A theoretical framework for sequential importance sampling and resampling," in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds., pp. 225–246, Springer, New York, NY, USA, January 2001.

[9] C. Berzuini, N. G. Best, W. R. Gilks, and C. Larizza, "Dynamic conditional independence models and Markov chain Monte Carlo methods," *Journal of the American Statistical Association*, vol. 92, no. 440, pp. 1403–1412, 1997.

[10] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *Journal of American Statistical Association*, vol. 89, no. 425, pp. 278–288, 1994.

[11] J. S. Liu and R. Chen, "Blind deconvolution via sequential imputations," *Journal of American Statistical Association*, vol. 90, no. 430, pp. 567–576, 1995.

[12] E. R. Beadle and P. M. Djurić, "A fast-weighted Bayesian bootstrap filter for nonlinear model state estimation," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 33, no. 1, pp. 338–343, 1997.

[13] D. Crisan, P. Del Moral, and T. J. Lyons, "Discrete filtering using branching and interacting particle systems," *Markov Processes and Related Fields*, vol. 5, no. 3, pp. 293–318, 1999.

[14] M. Bolić, P. M. Djurić, and S. Hong, "New resampling algorithms for particle filters," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP '03)*, vol. 2, pp. 589–592, Hong Kong, China, April 2003.

[15] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[16] J. S. Liu, R. Chen, and W. H. Wong, "Rejection control and sequential importance sampling," *Journal of American Statistical Association*, vol. 93, no. 443, pp. 1022–1031, 1998.

[17] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," submitted to *IEEE Trans. Signal Processing*.

[18] R. Chen, X. Wang, and J. S. Liu, "Adaptive joint detection and decoding in flat-fading channels via mixture Kalman filtering," *IEEE Transactions on Information Theory*, vol. 46, no. 6, pp. 2079–2094, 2000.

[19] P. M. Djurić, J. H. Kotecha, J. Zhang, et al., "Particle filtering," *IEEE Signal Processing Magazine*, vol. 20, no. 5, pp. 19–38, 2003.

[20] H. Zamiri-Jafarian and S. Pasupathy, "Adaptive MLSD receiver with identification of flat fading channels," in *Proc. IEEE Vehicular Technology Conference*, vol. 2, pp. 695–699, 1997.

Miodrag Bolić received the B.S. and M.S. degrees in electrical engineering and computer sciences (EECS) from the University of Belgrade, Yugoslavia, in 1996 and 2001, respectively. He is now a Ph.D. student at the State University of New York (SUNY), Stony Brook, USA, and he is working part-time for Symbol Technologies, Holtsville, NY. Before joining SUNY, he worked at the Institute of Nuclear Science Vinča, Yugoslavia. His research is related to the development and modification of the statistical signal processing algorithms for more efficient hardware implementation and VLSI architectures for digital signal processing.

Petar M. Djurić received his B.S. and M.S. degrees in electrical engineering from the University of Belgrade in 1981 and 1986, respectively, and his Ph.D. degree in electrical engineering from the University of Rhode Island in 1990. From 1981 to 1986, he was a Research Associate with the Institute of Nuclear Sciences, Vinča, Belgrade. Since 1990, he has been with State University of New York (SUNY), Stony Brook, where he is a Professor in the Department of Electrical and Computer Engineering. He works in the area of statistical signal processing, and his primary interests are in the theory of modeling, detection, estimation, and time series analysis and its application to a wide variety of disciplines including wireless communications and biomedicine.

Sangjin Hong received the B.S. and M.S. degrees in electrical engineering and computer sciences (EECS) from the University of California, Berkeley. He received his Ph.D. degree in EECS from the University of Michigan, Ann Arbor. He is currently with the Department of Electrical and Computer Engineering at State University of New York (SUNY), Stony Brook. Before joining SUNY, he worked at Ford Aerospace Corp. Computer Systems Division as a Systems Engineer. He also worked at Samsung Electronics in Korea as a Technical Consultant. His current research interests are in the areas of low-power VLSI design of multimedia wireless communications and digital signal processing systems, reconfigurable SoC design and optimization, VLSI signal processing, and low-complexity digital circuits. Prof. Hong served on numerous technical program committees for IEEE conferences. Prof. Hong is a Member of IEEE.