Tiancheng Li, Miodrag Bolić, and Petar M. Djurić

Resampling Methods for Particle Filtering

Classification, implementation, and strategies

Two decades ago, with the publication of [1], we witnessed the rebirth of particle filtering (PF) as a methodology for sequential signal processing. Since then, PF has become very popular because of its ability to process observations represented by nonlinear state-space models where the noises of the model can be non-Gaussian. This methodology has been adopted in various fields, including finance, geophysical systems, wireless communications, control, navigation and tracking, and robotics [2]. The popularity of PF has also spurred the publication of several review articles [2]–[6].

Using the PF method, we aim to track various distributions that arise in dynamic state-space models. The tracking is carried out by exploring the space of the states with randomly generated samples (also called *particles*). The distributions of interest are approximated by the generated particles as well as weights assigned to the particles.

There are many PF methods, and almost all of them are based on three operations: 1) particle propagation, 2) weight computation, and 3) resampling. Particle propagation and weight computation amount to the generation of particles and assignment of weights, whereas resampling replaces one set of particles and their weights with another set.

Particle generation and weight computation are computationally the most intensive steps. However, they are application dependent and can be easily implemented in parallel if parallel hardware is available. The resampling step is universal and generally state-dimension-free but is not naturally suitable for parallel processing. The resampling is essential for PF; without this step, PF will quickly produce a degenerate set of particles, i.e., a set in which a few particles dominate the rest of the particles with their weights. This means that the obtained estimates will be

Digital Object Identifier 10.1109/MSP.2014.2330626
Date of publication: 6 April 2015

1053-5888/15©2015IEEE

inaccurate and will have unacceptably large variances. With resampling, such deteriorations are prevented, which is why it is highly important to PF. Consequently, resampling has been extensively researched, and, as a result, various resampling schemes have been proposed [7]–[10].

Surveys of resampling methods can be found in [11]–[14]. These papers, however, only cover a small number of basic resampling methods and have become somewhat outdated. Furthermore, no classification has been made in these papers. This has been a disadvantage to researchers in getting a better grasp of the overall picture of resampling and of being able to readily choose the scheme that would fit their needs. This article aims to correct these shortfalls. More specifically, the main goal of this article is to introduce a new classification of resampling algorithms and provide a qualitative comparison of them. An additional goal is to set the grounds for further developments in resampling.

We divide the resampling algorithms into sequential and distributed/parallel algorithms. The most common are the sequential algorithms, where resampling is carried out from the approximating distribution using the latest weights. We also refer to these algorithms as *traditional*. However, researchers have explored other solutions, such as resampling that takes into account the history of the weights, resampling from approximate distributions, and resampling from only a part of the sampling space. In addition, approaches based on different ways of grouping particles have been proposed where grouping can be implemented, e.g., by using thresholds or by combining adjacent particles. These approaches might be of use when it is important to reduce the number of operations or to reduce communication between processing elements (PEs) in parallel implementations. We also discuss the frequency of resampling.

With multicore processors, the existence of general-purpose graphical processing units (GP-GPUs) in almost every computer and the emergence of embedded multicore and GP-GPU hardware, parallel processing can be readily implemented. The implementations of parallel resampling, however, can vary, which, in turn, reflect on their accuracy and speed. Some of the efforts in this area are described in this article.

In our view, future research efforts will be directed to specific implementations and theoretical analysis of the methods. The former include simplifying the resampling algorithms, development of better schemes with the ultimate goal of improving filtering performance, parallelization, and real-time implementations. The latter addresses the effects of the resampling algorithms on convergence and accuracy of approximation. As this article focuses mainly on providing guidelines to the readers and on qualitative description of the solutions, we do not explore issues such as robustness and do not provide theoretical proofs of any sort.

BACKGROUND OF PF AND RESAMPLING

A BRIEF REVIEW OF PF

We start with a brief review of PF and introduce the notation. There is a state-space model described by

$$x_t = g(x_{t-1}, u_t), \tag{1}$$
$$y_t = h(x_t, v_t), \tag{2}$$

where t is a time index and $t = 1, 2, \ldots$; $x_t \in R^{d_x}$ is the state of the model that is hidden (not observed); $y_t \in R^{d_y}$ is the observation; $u_t \in R^{d_u}$ and $v_t \in R^{d_v}$ are white noises that are independent of each other; and g: $R^{d_x} \times R^{d_u} \to R^{d_x}$ and h: $R^{d_x} \times R^{d_v} \to R^{d_y}$ are known functions. An alternative representation of these equations is by the probability distributions of the state, $p(x_t \,|\, x_{t-1})$, and of the observation, $p(y_t \,|\, x_t)$, which can be obtained from (1) and (2) and the probability distributions of u_t and v_t, respectively. The interest is in nonlinear models and where the noises in (1) and (2) are not necessarily Gaussian.

The objective of PF is the sequential estimation of distributions of the state, including the filtering distribution $p(x_t \,|\, y_{1:t})$, the predictive distribution $p(x_t \,|\, y_{1:t-1})$, or the smoothing distribution $p(x_t \,|\, y_{1:T})$, where $t < T$. Here, we focus on the filtering distribution. This distribution can be expressed in terms of the filtering distribution at time instant $t-1$, $p(x_{t-1} \,|\, y_{1:t-1})$, i.e., in a recursive form by

$$p(x_t \,|\, y_{1:t}) \;\propto\; \int p(y_t \,|\, x_t)\, p(x_t \,|\, x_{t-1})\, p(x_{t-1} \,|\, y_{1:t-1}) dx_{t-1}, \tag{3}$$

where the symbol $\propto$ signifies "proportional to." This update cannot be implemented analytically except in a very few cases, and, therefore, one resorts to approximations.

We reiterate that with PF, the underlying approximation is to represent continuous distributions by discrete random measures composed of particles $x_t^{(m)}$, which are possible values of the unknown state x_t and weights $w_t^{(m)}$ assigned to the particles. The distribution $p(x_{t-1} \,|\, y_{1:t-1})$ is approximated by a random measure of the form $\chi_{t-1} = \{x_{t-1}^{(m)},\ w_{t-1}^{(m)}\}_{m=1}^{M}$, where M is the number of particles, i.e.,

$$p(x_{t-1} \,|\, y_{1:t-1}) \approx \textstyle\sum_{m=1}^{M} w_{t-1}^{(m)}\, \delta\big(x_{t-1} - x_{t-1}^{(m)}\big), \tag{4}$$

where $\delta(\cdot)$ is the Dirac delta impulse and all the weights sum up to one. With this approximation, the integral in (3) can readily be solved, and we can write

$$p(x_t \,|\, y_{1:t}) \;\dot{\propto}\; p(y_t \,|\, x_t) \sum_{m=1}^{M} w_{t-1}^{(m)}\, p\big(x_t \,|\, x_{t-1}^{(m)}\big), \tag{5}$$

where $\dot{\propto}$ means "approximate proportionality."

The last expression shows how we can obtain the approximation χ_t of the filtering distribution recursively in time. At time instant $t-1$, one starts the construction of χ_t by generating particles $x_t^{(m)}$, which are used for representing $p(x_t \,|\, y_{1:t})$. This step of PF is referred to as *particle propagation* because a particle $x_{t-1}^{(m)}$ is moved forward in time and is a parent of $x_t^{(m)}$. For particle propagation and weight computation, we employ the concept of importance sampling [15]. Ideally, the propagated particles should be drawn from $p(x_t \,|\, y_{1:t})$, and then they will all have equal weights. However, this is infeasible in most cases, and,

therefore, one uses an instrumental function $\pi(x_t)$. For example, in [1], this function is $p(x_t|x_{t-1})$.

The second basic step of PF is the computation of the particle weights. To have correct inference from the generated particles, the theory shows that the generated particles from $\pi(x_t)$, which is different from $p(x_t|y_{1:t})$, have to be weighted [15]. Under mild conditions, one can show that these weights can be recursively computed according to

$$w_t^{(m)} \propto w_{t-1}^{(m)} \frac{p(y_t|x_t^{(m)})\, p\left(x_t^{(m)}|x_{t-1}^{(m)}\right)}{\pi(x_t^{(m)})}. \tag{6}$$

Often, the computation of the expression to the right of the proportionality sign is followed by normalization of the weights (so that they sum up to one).

Ideally, the weights of the particles should all be equal. On the other extreme, it is most undesirable if all the particles have weights equal to zero or one or a few particles have most of the weight and the rest of the particle weights are negligible. This is commonly called *degeneracy* and is exactly what eventually happens when PF is realized by using only the aforementioned two steps. Then, as the processing of the observations proceeds, the variance of the weights increases and reaches a point at which the random measure is a very poor approximation of the filtering distribution. For this reason, PF needs a third step, referred to as *resampling*.

WHEN IMPLEMENTING RESAMPLING, SEVERAL DECISIONS MUST BE MADE. THEY INCLUDE CHOOSING THE DISTRIBUTION FOR RESAMPLING, SPECIFYING THE SAMPLING STRATEGY, DETERMINING THE RESAMPLED SIZE, AND SELECTING THE FREQUENCY OF RESAMPLING.

THE BASICS OF RESAMPLING

With resampling, one aims to prevent the degeneracy of the propagated particles by modifying the random measure χ_t to $\tilde{\chi}_t$ and improving the exploration of the state space at $t+1$. While alleviating degeneracy during resampling, it is important that the random measure approximates the original distribution as well as possible and prevents bias in the estimates [16]. Although the approximation by $\tilde{\chi}_t$ is very similar to that of χ_t, the set of particles of $\tilde{\chi}_t$ is significantly different from that of χ_t. Resampling means that the particles from χ_t with large weights are more likely to dominate $\tilde{\chi}_t$ than particles with small weights, and, consequently, in the next time step, more new particles will be generated in the region of large weights. This is the reason for the improvement in exploration after resampling. The focus of exploration is shifted to the parts of the space with large probability masses. Because of resampling, the propagated particles from $\tilde{\chi}_t$ will have weights that are less discriminate than if the propagation was from the particles of χ_t. This is an intuitive idea with important practical and theoretical implications.

Formally, resampling is a process in which one samples from the original random measure $\chi_t = \{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}$ to create a new random measure $\tilde{\chi}_t = \{\tilde{x}_t^{(n)}, \tilde{w}_t^{(n)}\}_{n=1}^{N}$. Then, the random measure χ_t is replaced with $\tilde{\chi}_t$. In the process, some of the particles of χ_t are replicated, and, most likely, they are the particles with large weights. The particles of $\tilde{\chi}_t$ are used for propagation of new particles, and thus, they are the parents of $x_{t+1}^{(m)}$. We note that for the approximation of $p(x_t|y_{1:t})$, it is better to use χ_t than $\tilde{\chi}_t$. We also observe that the number of resampled particles N is not necessarily equal to the number of propagated particles. Traditional resampling methods keep them constant, and, usually, $M = N$. Finally, in most of the resampling methods, the weights of the particles after resampling are all equal.

Resampling, however, may introduce undesired effects. One of them is sample impoverishment. With resampling, low-weighted particles are most likely removed, and, thereby, the diversity of the particles is reduced [1], [9], [12]. For example, if a few particles of χ_t have most of the weight, many of the resampled particles will be the same, i.e., the number of different particles in $\tilde{\chi}_t$ will be small. The other effect is on the speedup of implementation of PF. We recall that PF is often used for signal processing when observations have to be processed in real time. A good solution for gaining in speed is to parallelize the PF. As will be shown, parallelizing the resampling is rather challenging.

The undesired effects of resampling have pushed researchers to investigate advanced methods for resampling. These methods have many features, including a variable number of particles, the removal of the restriction of equal weighting of the resampled particles, the avoidance of discarding low-weighted particles, and the introduction of parallel frameworks for resampling.

When implementing resampling, several decisions must be made. They include choosing the distribution for resampling, specifying the sampling strategy, determining the resampled size, and selecting the frequency of resampling.

CLASSIFICATION OF RESAMPLING SCHEMES

Our high-level classification of resampling methods with representative references is shown in Table 1. We first group the methods based on their implementation as sequential and parallel. We note that the parallel implementations represent two or more sequential implementations executed simultaneously. The sequential strategies are further classified based on whether the resampling is from a single distribution or from two or more distributions obtained from grouping of the particles (compound sampling). We also have a third category, referred to as *special strategies*. As the name suggests, the special strategies have features that separate them from single and compound sampling. Next, we make several remarks, of which the first four are also reflected in Table 1 (R1–R4).

[TABLE 1] THE HIGH-LEVEL CLASSIFICATION OF RESAMPLING METHODS.

CLASSIFICATIONS	R1 BASED ON THE DISTRIBUTION χ_t	R2 RESAMPLING OF ALL THE PARTICLES IN THE SAME WAY	R3 GROUPING	R4 USING INFORMATION FROM THE CURRENT TIME INSTANT
Sequential implementation	YES	YES/NO	YES/NO	YES/NO
■ SINGLE DISTRIBUTION SAMPLING [1], [7], [8], [10]	YES	YES	NO	YES
■ COMPOUND-SAMPLING		YES/NO FOR MANY: DIFFERENT OR NO RESAMPLING PER GROUP	BASED ON COMPOUND CRITERIA	
• THRESHOLDS/GROUPING-BASED RESAMPLING [11], [12], [24]				YES/NO FOR [11] AND [17]
• RESAMPLING THAT TAKES INTO ACCOUNT THE STATE [16]				YES
■ SPECIAL STRATEGIES		YES		
• MODIFIED RESAMPLING [11]	NO		NO	YES/NO FOR [18]
• VARIABLE-SIZE RESAMPLING [27]	YES			YES
• ROUGHENING [1]				
Parallel implementation			BASED ON ADJACENT PARTICLES	
■ MAPPING TO SPECIFIC HARDWARE PLATFORMS [31]–[33], [35]–[38] ■ DISTRIBUTED RESAMPLING [32], [40] ■ NORMALIZATION-FREE RESAMPLING [38], [39], [41]				

REMARK 1

One classification is based on the used distribution for resampling. Mainly, this is the distribution represented by χ_t. Other approaches include sampling from an approximate distribution. We will point out cases in which resampling is not performed from χ_t.

REMARK 2

Resampling can be performed in the same way on all of the particles, or it is possible to resample in different ways in different parts of the sampling space.

REMARK 3

There are methods that group the particles in some way before resampling is performed. In that respect, we distinguish among single-distribution sampling schemes (no grouping of particles), techniques that combine adjacent particles (which is the common approach in parallel implementations of resampling), and techniques that group particles for resampling based on some predefined criteria (referred to as *compound sampling*).

REMARK 4

Resampling can also be classified based on whether only particles from the current time step are involved in resampling, which is a common approach, the particles from previous time instants are considered [17], or some future particles are generated and taken into account for resampling [18]. Also, resampling can be classified based on whether only the weight is taken into account, which is the standard approach, or the state and the weight of particles are considered together [16].

REMARK 5

Resampling may be applied not in all but only in selected time steps. Compensations such as roughening [1] may be implemented with resampling for performance improvement.

REMARK 6

There are deterministic and stochastic resampling methods. The deterministic methods lead to a set of particles that are always the same for the same input set of particles.

IMPLEMENTATION OF RESAMPLING SCHEMES

In the following sections, we will explain various resampling methods and provide pseudocodes for selected algorithms. The pseudocodes are presented in a simple and unified way but not in forms that optimize the implementation of the algorithms. In practical implementations on a specific platform, programming techniques are required to maximally speed up the computation. For example, the cumulative sum calculation in code 1 can be implemented in MATLAB using vectors, which would be faster than the iterative calculation shown by code 1. Traditional sampling methods have already been described elsewhere; we present them here for completeness and because they are used in the compound, special, and parallel methods described later. We assume that the weights $w_t^{(m)}$ are normalized before resampling, i.e., $\sum_{m=1}^{M} w_t^{(m)} = 1$.

SINGLE-DISTRIBUTION SAMPLING METHODS

In this category, all the particles are resampled by using a single-distribution sampling procedure. The expected number of times $N_t^{(m)}$ that the mth particle is resampled is proportional to $w_t^{(m)}$, i.e.,

[TABLE 2] PSEUDOCODES OF MULTINOMIAL, STRATIFIED, SYSTEMATIC, RESIDUAL, BRANCH-KILL, AND ROUNDING-COPY RESAMPLING.

Code 1: Cumulative sum of normalized weights.

$[\{Q_t^{(m)}\}_{m=1}^{M}] = \text{CumulativeSum}[\{w_t^{(m)}\}_{m=1}^{M}]$

$Q_t^{(1)} = w_t^{(1)}$
FOR $m = 2{:}M$
  $Q_t^{(m)} = Q_t^{(m-1)} + w_t^{(m)}$
END

Code 2: Deterministic replication of particles.

$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N}, N] = \text{Replication}[\{x_t^{(m)}, N_t^{(m)}\}_{m=1}^{M}]$

$n = 0$
FOR $m = 1{:}M$
  FOR $h = 1{:}N_t^{(m)}$
    $n = n + 1$
    $\tilde{x}_t^{(n)} = x_t^{(m)}$
  END
END
$N = n$

Code 3: Multinomial/stratified/systematic resampling.

$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N}] = \text{Resample}[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, N]$

$[\{Q_t^{(m)}\}_{m=1}^{M}] = \text{CumulativeSum}[\{w_t^{(m)}\}_{m=1}^{M}]$
$n = 0$
/Systematic/stratified choice runs:
  $m = 1$
/Systematic choice runs
  $u_0 \sim U(0, 1/N]$
WHILE $(n \le N)$
/Stratified choice runs
  $u_0 \sim U(0, 1/N]$
/Systematic/stratified choice runs
  $u = u_0 + n/N$
/Multinomial choice runs
  $u \sim U(0, 1]$; $m = 1$
  WHILE $(Q_t^{(m)} < u)$
    $m = m + 1$
  END
  $n = n + 1$
  $\tilde{x}_t^{(n)} = x_t^{(m)}$
END

Code 4: Residual resampling/residual systematic resampling (RSR).

$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N}] = \text{Resample}[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, N]$

/RSR choice runs
  $\Delta u \sim U(0, 1/N]$
FOR $m = 1{:}M$
/Residual choice runs the following two lines
  $N_t^{(m)} = \text{Floor}(N \times w_t^{(m)})$
  $\hat{w}_t^{(m)} = w_t^{(m)} - N_t^{(m)}/N$
/RSR choice runs the following two lines
  $N_t^{(m)} = \text{Floor}(N \times (w_t^{(m)} - \Delta u)) + 1$
  $\Delta u = \Delta u + N_t^{(m)}/N - w_t^{(m)}$
END
$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N_t}, N_t] = \text{Replication}\ [\{x_t^{(m)}, N_t^{(m)}\}_{m=1}^{M}]$
/Residual choice runs the following four lines
  FOR $m = 1{:}M$
    $\hat{w}_t^{(m)} = \hat{w}_t^{(m)} \times N/(N - N_t)$
  END
  $[\{\tilde{x}_t^{(n)}\}_{n=N_t+1}^{N}] = \text{(Multinomial)Resample}[\{x_t^{(m)}, \hat{w}_t^{(m)}\}_{m=1}^{M}, N - N_t]$

Code 5: Branch-kill/rounding-copy resampling.

$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N}] = \text{Resample}[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, N_{\text{ref}}]$

FOR $m = 1{:}M$
/Branch kill choice runs the following five lines
  $u \sim U(0, 1/N_{\text{ref}}]$
  $N_t^{(m)} = \text{Floor}(N_{\text{ref}} \times w_t^{(m)})$
  IF $(N_{\text{ref}} \times w_t^{(m)} - N_t^{(m)}) \ge u$
    $N_t^{(m)} = N_t^{(m)} + 1$
  END
/Rounding-copy choice runs
  $N_t^{(m)} = \text{Round}(N_{\text{ref}} \times w_t^{(m)})$
END
$[\{\tilde{x}_t^{(n)}\}_{n=1}^{N}, N] = \text{Replication}[\{x_t^{(m)}, N_t^{(m)}\}_{m=1}^{M}]$

Note: The different methods are described with different colors, and the black text is common for all of them in each group, except where otherwise stated. Code 3 presents resampling methods based on the cumulative sum of the normalized weights (obtained by code 1). Codes 4 and 5 are methods that use deterministic replication (implemented by code 2). All of these codes produce resampled particles with equal weights.

$$E(N_t^{(m)} | w_t^{(m)}) = Nw_t^{(m)}. \tag{7}$$

This constraint is known as the *unbiasedness* or *proper-weighting condition* [10].

TRADITIONAL METHODS

Multinomial Resampling

The core idea of multinomial resampling [1] is to generate independently N random numbers, $u_t^{(n)}$ from the uniform distribution on (0, 1] and use them to select particles from χ_t. In the nth selection, the particle $x_t^{(m)}$ is chosen when the following condition is satisfied:

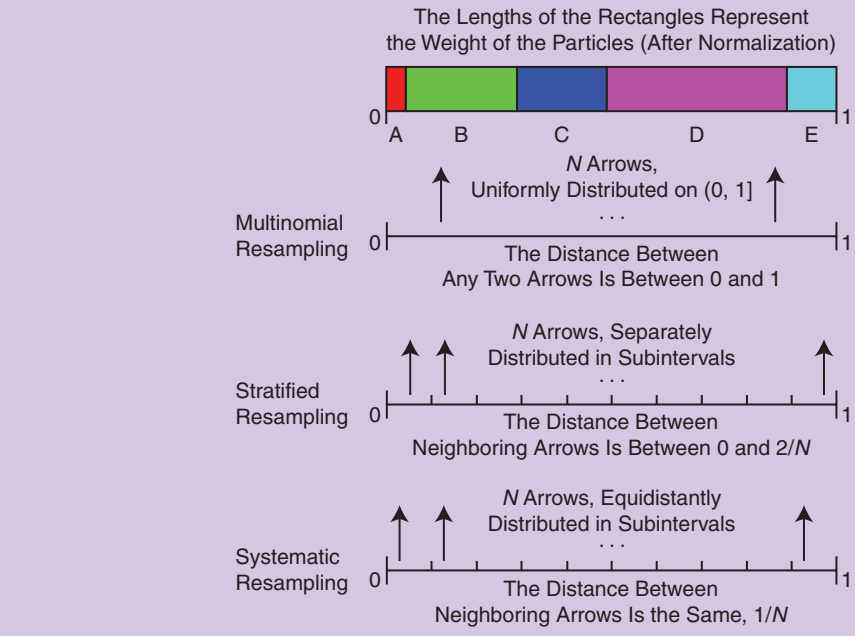
$$Q_t^{(m-1)} < u_t^{(n)} \le Q_t^{(m)}, \tag{8}$$

where

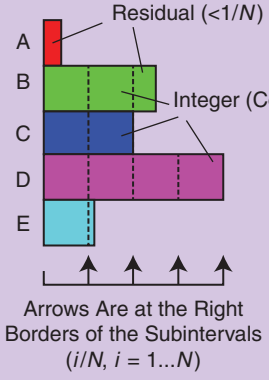
$$Q_t^{(m)} = \sum_{k=1}^{m} w_t^{(k)}. \tag{9}$$

Thus, the probability of selecting $x_t^{(m)}$ is the same as that of $u_t^{(n)}$ being in the interval bounded by the cumulative sum of the normalized weights as shown in (8). This sampling scheme satisfies the unbiasedness condition. The pseudocode of the cumulative sum of normalized weights is shown by code 1 in Table 2.

Multinomial resampling (see code 3 in Table 2) is also referred to as *simple random resampling*. Since the sampling of each particle is random, the upper and lower limits of the number of times a given particle is resampled are zero (not sampled) and N_t (sampled N_t times), respectively. This yields the maximum variance of the resampled particles.



(a)



Residual resampling, RSR, branch-kill, and rounding-copy replicate the particle based on the integer parts of each weight in the same way but deal with the residual parts differently.

Residual Resampling: Sample from the normalized fractions by using multinomial resampling (proceeded in a separate loop different to the integer replication).

RSR: Cumulatively add up residuals and sample them by using the idea of systematic resampling (proceeded in the same loop with the integer replication).

Branch-kill: If the residual $\geq u\sim(0, 1/N]$, take it as an integer and replicate it one time. Otherwise, abandon it.

Rounding-Copy: If the residual $\geq 1/(2N)$, take it as an integer and replicate it once. Otherwise, abandon it.

} Parallel Processing of Each Particle

} Do Not Preserve Constant Sample Size

(b)

[FIG1] A description of the traditional resampling methods. (a) A multinomial, stratified, and systematic resampling based on the cumulative sum of the normalized particle weights. The arrows represent sampling locations, and a particle is sampled if it is targeted by an arrow. (b) Methods based on residual resampling. They include two parts. The particles from the first part are obtained by replicating $x_t^{(m)}\lfloor Nw_t^{(m)}\rfloor$ times if $\lfloor Nw_t^{(m)}\rfloor \geq 1$, and the particles from the second part are generated based on the residual weights.

The computational complexity of multinomial resampling is of order $O(NM)$, where the M factor arises from the search of the required m in (8). It is known that multinomial resampling is not efficient [8], and this has motivated a search for faster methods. The binary search is explored to execute the search of m in (8), which reduces the computational complexity from M to $\log(M)$ [8]. The variance of the number of times a particle is resampled can be reduced by, for example, stratification and deterministic sampling.

Stratified/Systematic Resampling

Stratified resampling [7] divides the whole population of particles into subpopulations called *strata*. It prepartitions the $(0, 1]$ interval into N disjoint subintervals $(0, 1/N] \cup \cdots \cup (1-1/N, 1]$. The random numbers $\{u_t^{(n)}\}_{n=1}^N$ are drawn independently in each of these subintervals, i.e.,

$$u_t^{(n)} \sim U\Big(\frac{n-1}{N}, \frac{n}{N}\Big], \quad n = 1, 2, \ldots, N, \tag{10}$$

and then the bounding method based on the cumulative sum of normalized weights as shown in (8) is used.

Systematic resampling [7], [8] also exploits the idea of strata but in a different way. Now, $u_t^{(1)}$ is drawn from the uniform distribution on $(0, 1/N]$, and the rest of the u numbers are obtained deterministically, i.e.,

$$\begin{aligned} u_t^{(1)} &\sim U\Big(0, \frac{1}{N}\Big], \\ u_t^{(n)} &= u_t^{(1)} + \frac{n-1}{N}, \quad n = 2, 3, \ldots, N. \end{aligned} \tag{11}$$

Both the stratified and the systematic procedures are presented in Table 2. Their complexity is of order $O(N)$. Note that the systematic method is computationally more efficient than the stratified method because of the smaller number of random numbers that are generated. A visual description of the multinomial, stratified, and systematic resampling methods is displayed in Figure 1(a).

The upper and lower limits of the times the mth particle is resampled in the systematic method are $\lfloor N w_t^{(m)} \rfloor$ and $\lfloor N w_t^{(m)} \rfloor + 1$, respectively, where $\lfloor x \rfloor$ denotes the floor operation (the largest integer not exceeding x). By contrast, for stratified resampling, they are $\max(\lfloor N w_t^{(m)} \rfloor - 1, 0)$ and $\lfloor N w_t^{(m)} \rfloor + 2$, respectively, because the variables $\{u_t^{(n)}\}_{n=1}^{N}$ are not equidistant, and instead $\Delta u = u_t^{(n)} - u_t^{(n-1)}$ for $n = 2, 3, \ldots, N$ varies between 0 and $2/N$. When Δu is close to 0, a particle with a very small weight (close to 0 but bigger than Δu) can be resampled twice and when Δu is bigger than $2/N$, a particle with weight between $1/N$ and Δu can be discarded. This indicates that the variance of the number of a resampled particle by the systematic method is smaller than that of the stratified method.

Residual Resampling (Remainder Resampling)

Residual resampling [9], [10] consists of two stages. The first is a deterministic replication of each particle whose weight is bigger than $1/N$, and the second is random sampling using the remaining of the weights (referred to as *residuals*). The code for deterministic replication is shown by code 2, where $N_t^{(m)}$ represents the number of times the particle $x_t^{(m)}$ is replicated in this way. With residual resampling, the mth particle is resampled $N_t^{(m)} + R_t^{(m)}$ times, where $N_t^{(m)}$ and $R_t^{(m)}$ are the numbers of replications from the first and second stage, respectively, and where $N_t^{(m)} = \lfloor N w_t^{(m)} \rfloor$. The total number of replicated particles in the first stage is $N_t = \sum_{m=1}^{M} N_t^{(m)}$, and in the second stage, $R_t = N - N_t$. The residual of the weight is obtained from

$$\hat{w}_t^{(m)} = w_t^{(m)} - \frac{N_t^{(m)}}{N}. \tag{12}$$

In the second stage, the particles are drawn according to the residual weights and by using multinomial resampling (or another random sampling method), where the probability for selecting $x_t^{(m)}$ is proportional to the residual weight of that particle. Residual resampling has two loops taking on the order of $O(M) + O(R_t)$ time for computing.

The first stage represents a deterministic replication, and so the variation of the number of times a particle is resampled is only attributed to the second stage. Thus, the upper and lower limits of the number of times that the mth particle is resampled are $\lfloor N w_t^{(m)} \rfloor$ and $\lfloor N w_t^{(m)} \rfloor + R_t$, respectively, if the second stage is implemented using multinomial resampling. The code of the residual resampling method is given by code 4.

VARIATIONS OF TRADITIONAL RESAMPLING

The aforementioned four traditional methods are probably the best known and most used. They have been modified in various ways. For example, one of them removes the computationally expensive multinomial resampling part in residual resampling and implements resampling in a single loop (see code 4 in Table 2). The method is called *residual systematic resampling (RSR)* [12], [19]. RSR accumulates the fractional contributions of each particle in the searching sequence until it is large enough to generate a sample (which is equivalent to the accumulation idea used in systematic resampling). Then, no additional procedure is required for the residuals. Thus, one can have one iteration loop, and the complexity is of order $O(N)$.

If it is not mandatory to keep the particle size M constant at every time step and instead, the size is allowed to vary, we have simple ways of dealing with the particles in parallel and in one loop. We describe here two approaches. In the first approach, the number of replicated particles of $x_t^{(m)}$ is equal to $N_t^{(m)} = \lfloor N w_t^{(m)} \rfloor$ with probability $1 - p$ or equal to $N_t^{(m)} = \lfloor N w_t^{(m)} \rfloor + 1$ with probability p, where $p = N w_t^{(m)} - \lfloor N w_t^{(m)} \rfloor$. This method is called the *branch-kill* procedure [20] or *branching* [21]. In the second approach, $N_t^{(m)}$ is the nearest integer of $N w_t^{(m)}$, i.e., the rounding result of $N w_t^{(m)}$. We refer to this method as *rounding-copy* resampling [22]. Both methods (see code 5 in Table 2) require no additional operation and satisfy the unbiasedness condition but generate a varying sample size. The upper and lower limits of the number of replications of the mth particle in the branch-kill, rounding-copy, and RSR methods are all $\lfloor N w_t^{(m)} \rfloor$ and $\lfloor N w_t^{(m)} \rfloor + 1$, respectively.

[TABLE 3] A COMPARISON OF TRADITIONAL RESAMPLING METHODS (ALL THE RESAMPLED PARTICLES HAVE EQUAL WEIGHTS).

RESAMPLING METHODS	COMPUTATIONAL COMPLEXITY	ABLE TO KEEP CONSTANT SAMPLE SIZE	NUMBER OF RANDOM NUMBERS USED	NUMBER OF TIMES A PARTICLE IS REPLICATED	
				LOWER LIMIT	UPPER LIMIT
MULTINOMIAL	$O(NM)$ (OR $O(N \log M)$)	YES	N	0	N
RESIDUAL	$O(M) + O(R_t)$	YES	R_t	$\lfloor N w_t^{(m)} \rfloor$	$\lfloor N w_t^{(m)} \rfloor + R_t$
STRATIFIED	$O(N)$	YES	N	$\max(\lfloor N w_t^{(m)} \rfloor - 1, 0)$	$\lfloor N w_t^{(m)} \rfloor + 2$
SYSTEMATIC	$O(N)$	YES	1	$\lfloor N w_t^{(m)} \rfloor$	$\lfloor N w_t^{(m)} \rfloor + 1$
RESIDUAL SYSTEMATIC	$O(N)$	YES	1	$\lfloor N w_t^{(m)} \rfloor$	$\lfloor N w_t^{(m)} \rfloor + 1$
BRANCH-KILL	$O(N)$	NO	M	$\lfloor N w_t^{(m)} \rfloor$	$\lfloor N w_t^{(m)} \rfloor + 1$
ROUNDING-COPY	$O(N)$	NO	0	$\lfloor N w_t^{(m)} \rfloor$	$\lfloor N w_t^{(m)} \rfloor + 1$

A succinct comparison of the properties of the aforementioned methods is given in Table 3. The computational speeds of the methods are presented in [14] and [22].

COMPOUND SAMPLING

The resampling methods addressed so far are based on an approach where all the particles are sampled in the same way. This entails obtaining relatively similar resampling results. In all of the methods, the condition of unbiasedness is satisfied, and the resampled particles are equally weighted. In the following, we describe methods where resampling is realized without attempting to satisfy the conditions of unbiasedness and equal-weighting. This may entail risks of which the practitioner must be aware.

The compound sampling methods are based on grouping the particles by using predefined criteria prior to applying resampling. The groups are nonoverlapping, and they represent a partition of the whole particle population. The criterion for grouping is usually based on weight thresholds so that particles with similar weights are put in the same group, and then resampling is performed of each group in different ways. The reasons for applying compound resampling include decreasing the execution time and preserving particle diversity.

Compound resampling has its roots in stratified sampling. We will classify the methods based on whether the grouping is performed using a predefined threshold, which is either a constant or a function of the weights, or it is based on the particle values. We note that particles are often grouped when we implement parallel resampling. However, in parallel resampling, the groups are most commonly formed just by clustering index-neighboring particles.

THRESHOLD-/GROUPING-BASED RESAMPLING

In this category, particles are placed into different groups based on weight thresholds, and one uses different sampling strategies for each group to provide more flexibility for resampling. The threshold can be dynamic/adaptive or fixed, and there can be one or more thresholds.

Dynamic Threshold

The optimal resampling from [24] automatically sets a threshold value c_t, where c_t is a unique solution of

$$N = \sum_{m=1}^{M} \min\left(\frac{w_t^{(m)}}{c_t}, 1\right), \tag{13}$$

where N is given, and $N < M$. All the particles whose weights are above this threshold are entirely preserved rather than replicated. Thus, there are no multiple copies of these particles in the final set of N particles. The other particles are resampled with a probability corresponding to their weights and assigned a weight c_t. We see that the resampled particles do not have equal weights (see code 6). The main advantage of the method is that, among the unbiased resampling methods, it is optimal in terms of minimizing the squared error-loss function

$$E\left(\sum_{m=1}^{M} (\tilde{w}_t^{(m)} - w_t^{(m)})^2\right), \tag{14}$$

where $\tilde{w}_t^{(m)}$ is the new weight of $x_t^{(m)}$ when it is resampled; otherwise, $\tilde{w}_t^{(m)}$ is equal to zero. The method is appropriate for PF that uses increased number of propagated particles, and optimal resampling reduces the number to $N < M$. A disadvantage of the method is the need for calculating the value of c_t at each iteration. Also, the resampled particles may still suffer from degeneracy as the variance of the weights remains high.

Code 6: Optimal resampling.

$[\{\tilde{x}_t^{(n)}, \tilde{w}_t^{(n)}\}_{n=1}^{N}] = $ (Optimal)Resample $[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, N]$
Compute c_t satisfying (13)
$n = 0$; $h = 0$
FOR $m = 1: M$
  IF $w_t^{(m)} \geq c_t$
    $n = n + 1$
    $\tilde{x}_t^{(n)} = x_t^{(m)}$; $\tilde{w}_t^{(n)} = w_t^{(m)}$
  ELSE
    $h = h + 1$
    $A^{(h)} = x_t^{(m)}$; $B^{(h)} = w_t^{(m)}$
  END
END
$N_1 = n$
$[\{\tilde{x}_t^{(n)}\}_{n=N_1+1}^{N}] =$ (Stratified)Resample $[\{A^{(r)}, B^{(r)}\}_{r=1}^{h}, N - N_1]$
FOR $n = N_1 + 1: N$
  $\tilde{w}_t^{(n)} = c_t$
END

There are similarities between optimal resampling, rejection control (RC) [17] and partial RC (PRC) [11]. In RC, a control threshold c_t is computed, which may be a quantity given in advance, e.g., the median or a quantile of the weights, and the mth particle is accepted with a probability

$$p = \min\left(1, \frac{w_t^{(m)}}{c_t}\right). \tag{15}$$

In PRC, the particles with weights that are greater than or equal to c_t are automatically accepted, whereas other particles are accepted with probability p. This can be viewed as a combination of the rejection method and importance sampling. An accepted particle $x_t^{(m)}$ is reweighted with a new weight $\max(c_t, w_t^{(m)})$, and the rejected particles are replaced by particles regenerated from particles from previous time instances. These two forms of RC differ primarily in how far one goes back to regenerate particles. The RC does it to the earliest time, $t = 0$, while the PRC only regenerates particles from time instant $t - 1$ so that one saves on computations. These methods cannot be considered as candidates for real-time implementation because they have nondeterministic execution time and large memory requirements.

Fixed Threshold

A partial deterministic reallocation approach is proposed in [11] based on a fixed threshold, say $1/N$, where N is the desired sample size. The mth particle with a weight larger than $1/N$, is replicated $\lfloor N w_t^{(m)} \rfloor$ (or $\lfloor N w_t^{(m)} \rfloor + 1$) times, and the weights

after resampling are $w_t^{(m)}/\lfloor Nw_t^{(m)}\rfloor$ (or $w_t^{(m)}/(\lfloor Nw_t^{(m)}\rfloor+1)$) (this is referred to as *particle splitting* in the article). The mth particle with weight smaller than $1/N$ is sampled with probability $Nw_t^{(m)}$ and is weighted as $1/N$ (see code 7). The resampled particles are not equally weighted, and, notably, their weight sum is not one (a normalization step is additionally required). The sampling method is biased.

Code 7: Reallocation resampling.

$\left[\{\tilde{x}_t^{(n)}, \tilde{w}_t^{(n)}\}_{n=1}^{N^*}\right]$ = (Reallocation)Resample $[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, N]$
$n = 0$
FOR $m = 1{:}\ M$
  IF $w_t^{(m)} \geq 1/N$
    $N_t^{(m)}$ = Floor $(N \times w_t^{(m)})$ (or $N_t^{(m)}$ = Floor $(N \times w_t^{(m)}) + 1$)
    FOR $h = 1{:}\ N_t^{(m)}$
      $n = n + 1$
      $\tilde{x}_t^{(n)} = x_t^{(m)};\ \tilde{w}_t^{(n)} = w_t^{(m)}/N_t^{(m)}$
    END
  ELSE
    $u \sim U(0,\ 1/N]$
    IF $w_t^{(m)} \geq u$
      $n = n + 1$
      $\tilde{x}_t^{(n)} = x_t^{(m)};\ \tilde{w}_t^{(n)} = 1/N$
    END
  END
END
$N^* = n$

To reduce the complexity of hardware realization and the required power consumption, resampling is performed on only some of the particles. This is the idea behind partial resampling (PR) [12]. PR consists of two steps: in the first, the particles are classified as moderate, negligible, or dominating; and in the second, different resampling strategies are applied to each group of particles. In [17], three different resampling functions are proposed for determining which particles are resampled/discarded and how to allocate the weights, and, to that end, two thresholds are used. To further increase the processing speed, the classification of the particles can be overlapped with the weight computation (overlapped PR).

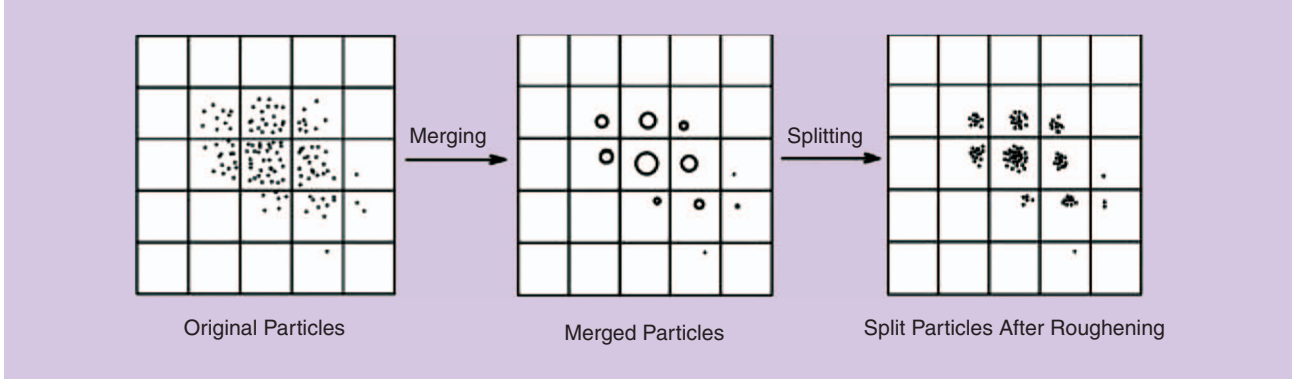
Grouping strategies may also be applied to alleviate the impoverishment that is practically inevitable in all the resampling methods presented so far. For example, the double systematic resampling approach from [25] partitions the particles into two groups, in low- and high-weighted particles and where the number of particles to be resampled from each group is specified. In this way, the low-weighted particles are resampled independently from the group of high-weighted particles. Obviously, this may lead to biased sampling.

There are several other methods from the literature that are built on similar ideas. Before their adoption, however, their features need to be better understood and firm guidelines for use provided.

RESAMPLING THAT TAKES INTO ACCOUNT PARTICLE VALUES

All of the above resampling methods are based on the particle weights. A possible way to improve resampling is to exploit the particle values (the state information they contain) during resampling. The particle distribution in the state space exhibits their diversity and, therefore, is the key for monitoring impoverishment. In this category of resampling methods, the particles are grouped with respect to not only their weights but also their values.

Particles in close proximity in values may represent a similar state, and, thus, they can be merged to reduce the number of particles with different values. This is the basis of deterministic resampling [16], which replaces the second stage of residual resampling by merging particles using their residuals. This aims at preserving the diversity of the particles so that no particles are discarded at all, which is helpful when the number of particles is small. Contrary to particle merging, particle splitting replaces a particle with a large weight (larger than a threshold) with a set of particles with the same values and whose sum of weights is equal to the original weight (see Figure 2). Particle merging is implemented before the weight-updating step to reduce the sample size to save computation, while particle splitting is applied after weight updating as an alternative to resampling to reduce the weight variance and to recover the sample size [23]. A main disadvantage of these methods is that the dimensionality-free property of resampling is destroyed.



[FIG2] Particle merging and splitting in a two-dimensional state space. The size of the circles represents the weight of particles.

Resampling Methods	Grouping and Sampling Methods Adopted					Benefits Reported	Disadvantages
	Sampling Methods (Particles with Low Weights)	Thresholds (Defining Strata)		Sampling Methods (Particles with High Weights)			
Optimal Resampling [24]	Stratified Sampling	Dynamic, c_t Where c_t Is a Unique Solution of (13)		Preserved		Minimized Squared Error-Loss Function	Additional Computation of (13); The Sample Size Decreases
(Partial) RC [17]	Replaces Particles by Regenerating Them from Previous Particles	Dynamic, c_t Where c_t May Be a Quantity Given in Advance or the Median or a Quantile of the Weights		Preserved and Reweighted with a New Weight (Reject Sampling)		Advantageous in Dealing with Sudden Changes in the Dynamic System	Increased Computational Intensity
Reallocation [11]	Rejection Sampling	1/*N*		Split		Reduced Resampling Variance (As Compared with Traditional Resampling Methods)	Varying Sample Size
PR [12]	(Small Weight) Preserved or Discarded	Fixed Threshold 1	(Moderate Weight) Preserved and Reweighted with Equal Weights	Fixed Threshold 2	(Significant Weight) Preserved and Reweighted with Equal or Different Weights	Reduced Computational Complexity	Thresholds Need to Be Carefully Specified
Deterministic Resampling [16]	Merged	1/*N*		Integer Parts (1/*N*) Are Replicated and Residuals Are Merged		Preserved Diversity of Particles (Without Discarding Any Particle)	Sensitive to the Dimensionality of the State
Double Systematic Resampling [25]	Systematic Resampling	1/*N*		Systematic Resampling		Increased Resampling Flexibility (Higher Probability for Drawing Small-Weighted Particles)	Two Independent Weight Normalization and Resampling

[FIG3] A comparison of several compound sampling methods.

COMPARISON OF SEVERAL COMPOUND SAMPLING METHODS

We emphasize that the core idea of compound sampling is to deal with particles from different groups differently. In Figure 3, we provide a succinct overview of different compound sampling methods, where the following terms for particles are used:

- *Discarded*: Particles that are discarded and not resampled.
- *Preserved*: Particles that are preserved with their weights being kept unless otherwise stated.
- *Merged*: Particles (in a specified space) that are merged to one particle with a state value equal to the weighted mean of the states of merged particles and with a weight equal to the sum of the original weights of the merged particles.
- *Split*: A particle that is divided into several copies. The split copies have the same state and weight, and their sum of weights is equal to the weight of the original particle.
- *Replicated*: A particle that is replicated, and each copy has the same value as the original particle as well as the same weight, unless otherwise stated.

Furthermore, in Table 4, we provide features of some of the compound sampling methods. They include optimal resampling [24], (partial) RC [17], reallocation [11], partial resampling [12], deterministic resampling [16], and double systematic resampling [25].

SPECIAL STRATEGIES

As previously mentioned, the key to combating degeneracy while avoiding impoverishment by resampling is the introduction of a compromise between concentration (the replication of large-weighted particles) and diversification (the discarding of negligible particles). To that end, several strategies have been proposed, including modified resampling, variable-size resampling, and roughening.

[TABLE 4] THE SPECIAL PROPERTIES OF SEVERAL COMPOUND RESAMPLING METHODS.

BIASED SAMPLING	UNEQUAL WEIGHTED	VARYING NUMBER OF PARTICLES	STATE CONSIDERED	UNIQUE/OUTSTANDING HIGHLIGHTS
[11], [12], [17], [25]	[11], [12], [16], [17], [24],	[11], [16], [24]	[16]	[11]: PARTICLE SPLITTING IS APPLIED; THE WEIGHT SUM OF THE RESAMPLED PARTICLES IS NOT ONE [12]: TWO (OR MORE) THRESHOLDS ARE USED [16]: AVOIDS DISCARDING ANY PARTICLE [17]: PREVIOUS PARTICLES ARE CONSIDERED; REJECT METHOD IS USED [24]: MINIMIZE THE SQUARED ERROR LOSS FUNCTION [25]: LOW-WEIGHTED PARTICLES ARE PROTECTED

In brief, the idea behind modified resampling is to draw particles from a distribution derived from the weights of the particles. The variable-size resampling, as the name suggests, provides a different number of samples with time so that a predefined criterion is satisfied (e.g., to reduce computational requirements, one uses a smaller number rather than a large number of particles, or to improve accuracy of tracking, one draws a larger number of particles). Roughening entails perturbing the locations of the resampled particles so that we reduce impoverishment, and, thus, it is performed once the resampling is completed. Next, we describe each of the strategies in more detail.

MODIFIED RESAMPLING

In generalized resampling [11], particles are resampled according to the probabilities $p_t^{(m)}$. The latter are usually equal to the particle weights $w_t^{(m)}$, but more generally, one can draw particles with probabilities that are functions of the weights, i.e.,

$$p_t^{(m)} \propto \left(w_t^{(m)}\right)^{\alpha}, \tag{16}$$

where $\alpha > 0$. When $0 < \alpha < 1$, the low-weighted particles get boosted, and the large-weighted particles have suppressed probabilities, and, thereby, the particle diversity improves. By contrast, $\alpha > 1$ entails increased preference for higher-weighted particles.

Knowledge about the next observation before resampling can be implemented via auxiliary weights [18]. In that way, the particles that are likely to have higher likelihoods have a better chance of surviving. There, the step of generating the auxiliary variable, which represents the fitness of the particle, can be viewed as a resampling step that takes into account both the immediate future and present states when carrying out selection. It is an appealing idea to fuse the information from the newest observations with the current weights while making a decision on the selection of particles.

VARIABLE-SIZE RESAMPLING

The use of a constant number of particles is not always the best choice because the complexity of the distributions of interest can vary drastically over time. In obtaining the number of particles that is both efficient and sufficient for approximating the distribution of interest, the underlying idea is to choose a small number of particles if the distribution is focused on a small part of the state space and to adopt a large number of particles if the distribution is much more spread out. This is the core rationale of the Kullback–Leibler divergence (KLD)-sampling approach [26], which determines the needed number of particles based on the KLD between the sample-based maximum likelihood estimate and a distribution of interest.

The required number N of particles can be determined so that, with the probability $1-\rho$, the KLD between the sample-based maximum likelihood estimate of a desired distribution and that distribution is less than a prespecified error bound threshold ε. In [26], it is found that

$$N = \frac{1}{2\varepsilon} q, \tag{17}$$

where

$$q = F^{-1}(1-\rho), \tag{18}$$

with $F^{-1}(\cdot)$ being the inverse of the cumulative chi-squared distribution with $k-1$ degrees of freedom, and k the number of bins [nonoverlapping (multi)dimensional intervals] used for sorting the particles. The value of N in (17) could be approximately computed [26]. In practice, the number of particles for resampling may be hard-limited to be not less than a minimum threshold.

Ideally, one would want to use as a desired distribution the posterior of the state. In [26], the posterior is approximated by the predictive distribution. Theoretically, it is more rigorous and flexible to apply (17) during resampling than in sampling, which leads to KLD-resampling [27]. There, the particles are resampled one by one until the required amount given by (17) is reached. Obviously, the disadvantage of the KLD-based method is that the particles need to be sorted out in bins defined on the state space, which can be very computationally costly.

ROUGHENING STRATEGIES

In obtaining an optimal set of particles, instead of designing the optimal proposal distribution, one can employ compensation. If impoverishment has already occurred after resampling, one approach to reducing it is to spread the overcentralized particles by roughening or jittering their values. This simply means that we add random noise (roughening noise) to the resampled particles. The roughening noise is normally Gaussian with zero mean and constant covariance. In [1], it is suggested that the noise covariance is diagonal with a standard deviation for a particular state component given by $\sigma = KDN^{-1/d_x}$, where D is the difference between the maximum and minimum values of the state component, K is a positive tuning constant chosen by the user, N is the number of particles, and d_x is the dimension of the state. The roughening may be applied 1) only at selected steps, 2) only to selected particles that are resampled from the same particle, and

[TABLE 5] PARALLEL PLATFORMS FOR PF IMPLEMENTATION.

PLATFORM	ADVANTAGES	DISADVANTAGES	APPLICATION
VLSI [31]	■ CUSTOMIZED ARCHITECTURE FOR SPECIFIC APPLICATION ■ VERY HIGH PROCESSING SPEED CAN BE ACHIEVED	■ VERY COSTLY ■ VERY LONG DESIGN CYCLES	■ NOT USED YET—SUITABLE FOR EXTREMELY LARGE QUANTITIES
FPGA [32]	■ CUSTOMIZED ARCHITECTURE FOR SPECIFIC APPLICATION	■ LIMITED FLOATING-POINT CAPABILITIES ■ LONG DESIGN CYCLE AND NEED TO KNOW HARDWARE AND HARDWARE DESIGN LANGUAGES	■ PROTOTYPING PLATFORM FOR REAL-TIME SYSTEMS
MULTICORE CENTRAL PROCESSING UNIT [33], [34]	■ COURSE-GRAINED PARALLELISM ON TWO TO EIGHT CORES. RELATIVELY EASY TO ACCESS AND TO PROGRAM	■ LARGE AMOUNT OF SEQUENTIAL PROCESSING	■ ACCELERATING SIMULATIONS
GP-GPU [35]–[37]	■ FINE-GRAINED PARALLELISM CAN BE ACHIEVED ■ AVAILABLE ON ALMOST EVERY COMPUTER ■ EASY TO PROGRAM	■ KNOWLEDGE OF GPU PROGRAMMING LANGUAGES SUCH AS CUDA OR OPENCL	■ ACCELERATING SIMULATIONS AND SOME HIGH-PERFORMANCE REAL-TIME APPLICATIONS
EMBEDDED GPU	■ PROGRAM IN THE SAME WAYS AS GP-GPU ■ CANDIDATE FOR PF FOR EMBEDDED SYSTEMS	■ LESS PARALLELISM AVAILABLE THAN IN GP-GPU ■ KNOWLEDGE OF GPU PROGRAMMING LANGUAGES	■ NEW PLATFORMS FOR EMBEDDED SYSTEMS SUITABLE TO REAL-TIME PERFORMANCE
CLUSTER AND MULTICOMPUTERS [33], [39]	■ COMPLEX PARTICLE FILTERS CAN BE SIGNIFICANTLY ACCELERATED	■ KNOWLEDGE OF PARALLEL PROGRAMMING LANGUAGES SUCH AS OPENMP OR MESSAGE PASSING INFERENCE	■ ACCELERATING SIMULATIONS

3) only in a few dimensions of the state space. Similar ideas for diversification can be implemented by using Markov chain Monte Carlo (MCMC) sampling. The so-called resample-move algorithm from [28] has a move step after the resampling step based on MCMC sampling. The move step performs one or more iterations on each of the resampled particles, thereby rejuvenating the diversity of particles.

An important alternative to the roughening strategy is the use of Gaussian kernels. With the kernels, we smooth the posterior density by convolving each particle with a diffusion kernel. This is also known as *regularization*, and it amounts to replacing the discrete distribution defined by the particles and their weights with a continuous approximation [29]. The resampling step is then changed to simulations from the continuous distribution, which generates a new particle set. All of these approaches produce diversified copies of the same particles.

PF OFTEN SUFFERS FROM A HEAVY COMPUTATIONAL COST, ESPECIALLY WHEN THE DIMENSION OF THE STATE-SPACE MODEL IS HIGH AND THE NUMBER OF USED PARTICLES HAS TO BE LARGE.

PARALLEL PROCESSING

Despite its successful applications, PF often suffers from a heavy computational cost, especially when the dimension of the state-space model is high and the number of used particles has to be large. However, the execution of PF can be accelerated through parallelization for both offline and real-time processing. An important area where parallelized PF also takes place is in signal processing over networks, where PF operates in parallel at the nodes of the network [30]. There, the problems of PF are of a different nature than the ones we address here. In this article, we focus on describing the parallelization of sequential PF and the resampling algorithm.

PARALLEL PROCESSING ARCHITECTURES FOR PF

Parallel hardware devices where PF has been implemented and reported in the literature include custom very large-scale integration (VLSI) chips [31], field-programmable gate arrays (FPGAs) [32], multicore processors [33], [34], GP-GPUs [35]–[38], and computer clusters/multicomputers [33], [39]. The advantages, disadvantages, and potential applications of these platforms for PF implementation are shown in Table 5. Hardware implementations on VLSI and FPGA result in high-speed implementation since every mathematical function is customized and implemented in hardware. However, VLSI design is very expensive, and, to the best of our knowledge, there are no commercial chips with implemented PF yet. The number of floating point units that can be executed in parallel on an FPGA is smaller than on state-of-the-art GP-GPUs. Also, floating-to-fixed-point conversion is very complicated, and particle filters require a large number of bits in fixed-point representation. In addition, programming of GPUs is simpler than designing hardware for FPGAs. Multicore platforms and GP-GPUs are now available on almost every computer so that lately we have been witnessing more parallel PF implementations on GP-GPUs.

Recently, new chips with embedded multiple processors and GP-GPUs have appeared from various companies. They support the same programming models as GP-GPUs, including CUDA and Open Computing Language (OpenCL). They are intended for high-performance embedded applications and have much lower power consumption than GP systems. Furthermore, many-core chips that have more than 16 processors on a single chip have appeared and have started to be used for high-performance

embedded applications. As far as we know, PF has not yet been implemented on these embedded platforms.

CHALLENGES

The particle generation and weight updating are computationally the most intensive steps and can be implemented in parallel if parallel processing hardware is available. Resampling, normalization, and computing estimates are inherently sequential, and they limit the speedup that can be achieved with parallel processing. An additional step, called *particle allocation*, is usually needed for efficient parallel implementation. We describe this step with an example. Assume that parallel processing is performed using a generic architecture with three PEs. Let each PE perform sampling and weight computation of an equal number of particles equal to $M/3$, where $M/3$ is an integer. After resampling, some particles are replicated, and some are discarded. Now consider a situation where all the particles in PE1 and PE3 are discarded, and the majority of the particles in PE2 are replicated. When the next observation comes, there will be no particles in PE1 and PE3 for processing, and almost all the processing will occur sequentially in PE2, resulting in low PE utilization and load imbalance. To avoid this situation, particles from PE2 need to be redistributed (allocated) to the other PEs so that all the PEs have an equal amount of particles before the next step. The particle allocation can be done by sending particles and their weights in message-passing architectures or by reallocating particles (or their indexes) in the global memory in shared-memory architectures. Particle allocation requires additional time, and, consequently, it affects the speedup.

Major challenges for the parallel implementation of PF include:

- There might be computationally complex operations in the sampling and weight-computation steps, including random number generation. Some of these operations might not be supported by all processing architectures.
- Operations such as normalization, computation of output estimates, and computation of cumulative sums in some resampling algorithms are sequential and require particles from all PEs.
- Particle allocation requires frequent communication with the global shared memory in shared-memory systems, which is a bottleneck. Similarly, the communication between the PEs in message-passing systems consumes significant time.
- Platform-specific challenges are related to the fact that every platform requires modifications or implementations that are platform specific. For example, design of a hardware random number generator is required in FPGA platforms, while an efficient parallel software algorithm for random number generation is needed for GPUs.

PERFORMANCE METRICS AND TERMINOLOGY

The major objective of parallel implementation is to reduce the execution time, and the most commonly used metric is speedup [34], [35]. We define the execution time of particle filters as the time they need to process one observation [31], [32], [36]. The speedup is defined as the ratio of the execution time of the best possible serial algorithm (on a single processor) to the execution time of the chosen algorithm on a parallel system based on multiple processors. The efficiency is defined as the speedup divided by the number of processors [33]. System utilization indicates the percentage of resources that was kept busy during the execution of a parallel program. Communication overhead is defined as the percentage of time spent on interprocess communication and all noncomputing operations [33]. The degree of parallelism is a measure of the numbers of threads of computation that can be carried out simultaneously. Data dependencies are the result of precedence constraints between operations, and they prevent concurrent execution. For example, the degree of parallelism for the weight-computation step is equal to the number of particles. A number of papers have been devoted to increasing the degree of parallelism and system utilization during resampling to decrease execution time.

IF IMPOVERISHMENT HAS ALREADY OCCURRED AFTER RESAMPLING, ONE APPROACH TO REDUCING IT IS TO SPREAD THE OVERCENTRALIZED PARTICLES BY ROUGHENING OR JITTERING THEIR VALUES.

Other types of performance metrics are related to tracking the performance of the parallel particle filter. Researchers mainly rely on examples to monitor the change of the mean square error [32], [34] or the effective sample size [33], [40] of the parallel versus sequential implementation of PF. Theoretical work on the performance or loss of performance of PF due to parallel implementation is still missing. One attempt is made in [45], where the analysis of the sample variance of the weights, the distortion of the probability measure, and the variance of estimators of a distributed PF are given. In [33], the mean and standard deviation of the log likelihood are calculated to validate that the resampling procedure does not affect the performance of the particle filter.

DISTRIBUTED RESAMPLING

Much work has been focused on parallelization of traditional resampling, referred to as *distributed resampling*. Two main classes of approaches have been applied: 1) the algorithm is modified in a way that the result of resampling is not changed in comparison with the sequential algorithm, and 2) the algorithm is modified, and it has been shown that, mainly through simulations, its parallel implementation provides similar results to the original sequential resampling. The majority of the algorithmic modifications is performed to reduce the communication overhead mainly during the particle allocation step.

An example of an algorithm that does not change the result of standard resampling is distributed resampling with proportional allocation [32]. Here, the resampling step is divided into two steps. In the first step, each PE is considered as a particle whose weight is the cumulative weight of all the particles of this PE. The resampling is performed only on the cumulative weights to determine how many particles each PE needs to generate. Next, local resampling is performed in parallel in each PE. In this way, the degree of

parallelism for the resampling step is equal to the number of PEs. Subsequently, the particle allocation step is performed, in which the excess of particles after resampling from some PEs is communicated or made available to the PEs with the deficiency of particles. Central (sequential) processing is still needed for the first step of resampling, normalization, and computing the output estimate. A number of variations of this method have been devised where load unbalance and the amount of central processing are reduced. The load balancing solution for the GP-GPU platform has been proposed in [36] using CUDA. All the particles are stored in a global GP-GPU memory, which is a shared memory, and writing to it is slow. During the particle-allocation step, the same replicated particles are written many times to the memory. After the particle allocation, each set of particles from the global memory is apportioned to different parallel threads. The optimization performed here is architecture (GP-GPU) specific.

SERIAL COMPUTATION AND GLOBAL COMMUNICATION ARE INEVITABLE IN WEIGHT NORMALIZATION, AND, THEREFORE, ALTERNATIVES THAT ARE FREE OF NORMALIZATION BECOME ATTRACTIVE.

The aforementioned methods require some central (sequential) processing to perform resampling. To further improve the speed of resampling, it is desirable to perform local particle allocation without having a central unit. Localized particle allocation produces different results than traditional resampling algorithms, but it reduces communication overhead, making it concurrent. In localized resampling, only a subset of the particles is exchanged between neighboring PEs/threads, and the exchange is preferably performed in parallel. The resampling and all the other operations are also performed in parallel. Various solutions have been proposed that depend on different parameters, including the number of PEs, the number of particles to be exchanged, the types of particles to be exchanged, and the selection of PEs where the particles are sent. Resampling with nonproportional allocation (RNA) [32] allows for each PE to perform local resampling and to exchange some fixed number of particles with one neighboring PE. The platform where PF was implemented in [32] was an FPGA. RNA-based implementation of parallel resampling is carried out in one of the first implementations of PF on GP-GPU [37]. In other implementations, one or several of the largest particles are exchanged with neighboring PEs. As an example, [38] presents a GP-GPU implementation where local sorting is performed to order the particles that are being exchanged. In [40], a method is proposed for finding the optimal portion of particles for exchange between adjacent PEs and for achieving the mixing of the posterior distributions among the adjacent PEs, so as to preserve particle diversity.

NORMALIZATION-FREE RESAMPLING

Serial computation and global communication are inevitable in weight normalization, and, therefore, alternatives that are free of normalization become attractive. In this category, resampling is performed free of weight normalization, differing from all the resampling methods presented so far. In brief, the particles are (re-) sampled based on their relative magnitude of the (nonnormalized) weights, e.g., the ratios between the weights globally, or the absolute weight comparison between local neighboring (two or three) particles. The former is globally unbiased sampling, while the latter is local sampling that is almost surely biased.

Ratios Between Weights

Metropolis [38] and independent Metropolis–Hastings sampling [39] require only ratios between weights that do not need to be normalized and therefore threads can process in parallel, see code 8, for example. The solution presented in [38] addresses GP-GPU implementation where the amount of parallel computational resources is abundant, and it stresses that even though Metropolis sampling is more computationally intensive than traditional resampling algorithms, it is as fast on a GP-GPU because there are no dependent operations on the weights. There, sampling from $U(1,\ldots M)$ returns a value randomly selected from the set $\{1,\ldots,M\}$.

Code 8 is an iterative process of sampling based on constructing a Markov chain. More specifically, it uses a Metropolis–Hastings move step for searching in a particle set for a particle with a large weight to replace the current particle. The depth of the search (burn-in) is denoted by B. It is desirable that the number of times a particle is sampled is proportional to its weight. As in most MCMC algorithms, deeper searching will provide better results (closer to the desired distribution).

The selection of B is a tradeoff between speed and reliability. While runtime is reduced with fewer steps, the sample will be biased if B is too small to ensure convergence. Similarly, a number of additional particles for burn-in (the time during which the Markov chain is in a transient state) is needed. The particles that are generated in the burn-in period are discarded. A disadvantage of this method is that, in most cases, it is difficult to estimate the required burn-in period.

Code 8: Metropolis resampling.

$[\{\tilde{x}_t^{(m)}, \tilde{w}_t^{(m)}\}_{m=1}^{M}] =$ (Metropolis)Resample $[\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}, B]$
FOR $m = 1$: M (can be in parallel)
  $k = m$
  FOR $n = 1$: B
    $u \sim U(0,1]$; $l \sim U\{1,\ldots,M\}$
    IF $u \le w_t^{(l)}/w_t^{(k)}$
      $k = l$
    END
  END
  $\tilde{x}_t^{(m)} = x_t^{(k)}$; $\tilde{w}_t^{(m)} = 1/M$
END

Local Neighbor-Comparison Methods

Now, we consider more straightforward local methods that include local Monte Carlo [10] and Local Selection (LS) methods [41]. The

local methods consist of randomly sampling a particle from a small set of successive particles, where the sampling is based only on the weights of the particles in the set. The resampled particles preserve their weights. Assume, for simplicity, that the number of PEs K is equal to the number of particles M (if $K < M$, particles allocated to the same PE are processed serially), and the mth PE contains a particle $\{x_t^{(m)}, w_t^{(m)}\}$ in its memory, where $w_t^{(m)}$ is not normalized. Before the parallel resampling, a communication step is required, where the mth PE transmits its particle to its two neighboring PEs. Then, each PE contains three particles, and the LS resampling can be realized as by code 9.

For consistency of description, note that $w_t^{(0)}$ refers to $w_t^{(M)}$, while $w_t^{(M+1)}$ refers to $w_t^{(1)}$.

Obviously, the number of replications of each particle in LS is very limited in local groups and is maximally three. This can cause a biased result, especially if high-weighted particles are next to each other. Then, some of them will be discarded and the performance will be degraded.

Code 9: Local selection resampling.

$[\{\tilde{x}_t^{(m)}, \tilde{w}_t^{(m)}\}_{m=1}^{M}] = (\text{LS})\text{Resample}\ [\{x_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}]$
FOR $m = 1{:}\ M$ (in parallel)
  $W = w_t^{(m-1)} + w_t^{(m)} + w_t^{(m+1)}$
  $T_1 = w_t^{(m-1)}/W$
  $T_2 = (w_t^{(m-1)} + w_t^{(m)})/W$
  $u \sim U(0, 1]$
  IF $u \le T_1$
    $\tilde{x}_t^{(m)} = x_t^{(m-1)};\ \tilde{w}_t^{(m)} = w_t^{(m-1)}$
  ELSE IF $T_1 < u \le T_2$
    $\tilde{x}_t^{(m)} = x_t^{(m)};\ \tilde{w}_t^{(m)} = w_t^{(m)}$
  ELSE
    $\tilde{x}_t^{(m)} = x_t^{(m+1)};\ \tilde{w}_t^{(m)} = w_t^{(m+1)}$
  END
END

We note that although weight normalization is avoided in these normalization-free resampling approaches, it is still necessary for computing the filtering estimates.

OTHER TYPES OF PARALLELIZATION OF PF

We reiterate that the resampling process represents a bottleneck in parallel implementations. For this reason, there have been efforts to develop PF methods that do not require resampling [see first item below] and overlap the operations of multiple PF or the steps of the same PF so that all the operations are sequential but are executed concurrently (second and fourth items below). More specifically, these efforts include the following:

1) Removing the resampling step from PF, such as in nonresampling PF detector [42] and Gaussian PF (GPF) [43], which are resampling-free.

2) Running several PFs independently on separate processors or a set of agents in a distributed network [30]. Some or all of the agents perform local PF and interact with other agents to calculate a global state estimate. These decentralized agent networks do not include a central unit.

3) Decomposing the state into two parts and considering the filtering problem as two nested subproblems. These two problems are then handled by separate PFs. This is also referred to as *decentralized PF* [44].

4) Pipelining the sampling and resampling: when a particle is resampled, it processes the sampling (particle propagation and weight updating) ahead that does not need to wait for other particles, i.e., the sampling in the next iteration will be produced before the resampling is finished.

THE RESAMPLING PROCESS REPRESENTS A BOTTLENECK IN PARALLEL IMPLEMENTATIONS.

Before concluding the subsection on parallel processing, we list some topics for further research on it. They include:

- Applying PF to real-world applications by implementing them on embedded multicore, embedded GPUs, and many-core chips and guaranteeing real-time performance (for example, current GPUs do not incorporate hard real-time features).
- Deriving optimization criteria that allow for evaluation of the quality of the localized distributed resampling algorithms versus sequential traditional resampling algorithms. Evaluating the practical benefit of parallel processing in terms of not only computing speed but also filtering accuracy. Some initial comparisons of parallel implementations of resampling that offer theoretical analysis and simulations are available. For example, analysis of resampling via RNA and LS in terms of reduction of the sample variance of the weights, the distortion of the probability measure, and the variance of estimators is given in [45].
- Devising new parallel algorithms for specific architectures—for example, communication is very expensive in GPGPUs, and therefore, one research direction is deriving algorithms with reduced communication between the PEs that can even be more computationally intensive.

FREQUENCY OF RESAMPLING

The benefits of resampling are accompanied with potential side effects such as sample impoverishment and prevention of parallel processing as explained above. Thus, resampling should only be applied when necessary and therefore it is important to have a method for determining how frequently or when to implement resampling. Two schedules for resampling have been proposed: deterministic and adaptive. In a deterministic schedule, one does resampling at fixed times $t_1, t_2, \ldots$, where t_i is often chosen to be $i \times t_1$. In an adaptive schedule, the times at which resampling occurs are selected by checking a criterion that assesses the quality of the current weights. In that case, whenever the criterion for quality is below a given threshold, the resampling step is triggered. This tends to produce better performance of PF than deterministic scheduling due to its flexibility.

As a key to adaptive schedules, the criterion for implementing resampling is usually based on the variation of the weights,

which reflects the degree of weight degeneracy. One such criterion is the effective sample size (ESS) N_{eff} defined by [46]

$$N_{\text{eff},t} = \frac{M}{1 + \text{Var}(w_t^*)}, \tag{19}$$

where w_t^* is a nonnormalized weight, and the variance is computed with respect to the sampling distribution. Typically, obtaining N_{eff} from (19) is impossible [46]. Instead, one may employ the rule of thumb estimate given by

$$N_{\text{eff},t} = \left(\sum_{m=1}^{M} \left(w_t^{(m)}\right)^2\right)^{-1}. \tag{20}$$

An application of the Cauchy–Schwarz inequality leads to the (intuitive) conclusion that $N_{\text{eff},t} \leq M$. It is also clear that $1 \leq N_{\text{eff},t} \leq M$. Resampling occurs when the ESS falls below a selected threshold, γ_r. If γ_r is set to $\gamma_r = 0$, resampling never takes place, and $\gamma_r = M$, meaning that resampling occurs at every time step. Several different criteria to calculate the ESS have also been proposed; see, e.g., [8], [47], and [48]. It is necessary to note that criteria based on (20) are primarily used in tracking low-dimensional states. In high-dimensional problems, other metrics may be more appropriate.

CONCLUSIONS

In this article, the state of the art of resampling methods was reviewed. The methods were classified and their properties were compared in the framework of the proposed classifications. The emphasis in the article was on the classification and qualitative descriptions of the algorithms. The intention was to provide guidelines to practitioners and researchers.

Some final comments:

- The resampling methods can hardly output much different results if they satisfy the unbiasedness condition, preserve a constant number of particles, and equally weight the resampled particles.
- If these restrictions are removed, some benefits may be obtained, e.g., adaptive adjustment of the number of particles, preservation of particle diversity, and alleviation of impoverishment.
- Compound sampling and special strategies, such as modified resampling and variable-size resampling, and compensations after resampling provide more flexibility. They balance the necessity for diversity and the need for concentration that lies in the center of sample degeneracy and impoverishment. They may offer better approximation and benefits in practice, but often at the price of higher computational costs.
- The normalization of weights and load imbalance of particles after resampling are main barriers for parallelization of resampling. Ways to carry out normalization, to output filter estimate, and to manage communication between PEs distinguish existing parallel PF algorithms.
- An issue that we have not discussed is the theoretical effects of resampling on the convergence of the PF estimates. The resampling step is crucial for uniform convergence results, and some recent theoretical results on this and related to deterministic and random resampling can be found in [49].
- The research on the resampling of particle filters is going in multiple directions:
  - implementation specific
    - simplifying resampling algorithms for real-time implementation
    - adjustment of the algorithms to specific hardware/processing architectures
    - parallelization of the resampling
    - application and acceleration of resampling to non–PF-based problems such as importance sampling and the forward-backward algorithm [50]
  - theoretical analysis
    - analysis of the features of resampling algorithms without considering them as a part of PF
    - analysis of the effects of different resampling algorithms on the PF convergence and accuracy of tracking.

ACKNOWLEDGMENTS

The work of Tiancheng Li has been supported by the Excellent Doctorate Foundation of Northwestern Polytechnical University and by the National Natural Science Foundation of China (under Award 51475383). The work of Petar M. Djurić has been supported by the NSF under Award CCF-1320626.

AUTHORS

Tiancheng Li (t.c.li@mail.nwpu.edu.cn) received his bachelor's degree in mechanical and electrical engineering, with a minor in business administration, from Harbin Engineering University, China, in 2008, and his Ph.D. degree in electrical and electronic engineering from London South Bank University, United Kingdom, in 2013. He is a currently a Ph.D. candidate in mechatronics engineering at Northwestern Polytechnical University, China, and a postdoctoral associate with the Bioinformatic, Intelligent Systems, and Educational Technology Group, University of Salamanca, Spain. His research interests are in the general area of statistical signal processing, information fusion, system modeling with a particular focus on particle filtering, finite set statistics, and multiple object tracking.

Miodrag Bolić (mbolic@eecs.uottawa.ca) received his B.Sc. and M.Sc. degrees in electrical engineering from the University of Belgrade, Serbia, in 1996 and 2001, respectively, and his Ph.D. degree in electrical engineering from Stony Brook University, New York, in 2004. Since 2004, he has been with the University of Ottawa, Canada, where he is an associate professor with the School of Electrical Engineering and Computer Science. His research interests include signal processing architectures, hardware/software accelerators, biomedical signal processing and instrumentation, and the Internet of Things. He is director of the Computer Architecture, Radio-Frequency Identification, and Medical Devices Research Groups at the University of Ottawa. He is a Senior Member of the IEEE.

Petar M. Djurić (petar.djuric@stonybrook.edu) received his B.S. and M.S. degrees in electrical engineering from the

University of Belgrade, Serbia, and his Ph.D. degree in electrical engineering from the University of Rhode Island. He is currently a professor in the Department of Electrical and Computer Engineering at Stony Brook University, New York. His research has been in signal and information processing, with an emphasis on Monte Carlo-based methods, signal processing over networks and applications in wireless sensor networks, and radio-frequency identification. He received the IEEE Signal Processing Magazine Best Paper Award in 2007 and the EURASIP Technical Achievement Award in 2012. He is a Fellow of the IEEE.

REFERENCES

[1] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc., F Radar Signal Process*., vol. 140, no. 2, pp. 107–113, 1993.

[2] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," in *Handbook of Nonlinear Filtering*, E. D. Crisan and B. Rozovsky, Eds. Oxford, London: Oxford Univ. Press, 2011, pp. 656–704.

[3] A. Doucet, S. J. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Stat. Comput.*, vol. 10, no. 3, pp. 197–208, 2000.

[4] M. 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.

[5] P. M. Djurić, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, and J. Míguez, "Particle filtering," *IEEE Signal Processing Mag.*, vol. 20, no. 5, pp. 19–38, 2003.

[6] O. Cappé, S. J. Godsill, and E. Moulines, "An overview of existing methods and recent advances in sequential Monte Carlo," *IEEE Proc.*, vol. 95, no. 5, pp. 899–924, 2007.

[7] G. Kitagawa, "Monte Carlo filter and smoother and non-Gaussian nonlinear state space models," *J. Comput. Graph. Stat.*, vol. 5, no. 1, pp. 1–25, 1996.

[8] J. Carpenter, P. Clifford, and P. Fearnhead, "An improved particle filter for nonlinear problems," *IEE Proc., Radar Sonar Navigat.*, vol. 146, no. 1, pp. 2–7, 1999.

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

[10] J. Liu and R. Chen, "Sequential Monte-Carlo methods for dynamic systems," *J. Amer. Statist. Assoc.*, vol. 93, no. 443, pp. 1032–1044, 1998.

[11] 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. New York: Springer, 2001, pp. 225–246.

[12] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP J. Appl. Signal Process.*, vol. 2004, no. 15, pp. 2267–2277, 2004.

[13] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *Proc. 4th Int. Symp. Image and Signal Processing and Analysis*, 2005, pp. 64–69.

[14] J. D. Hol, T. B. Schön, and F. Gustafsson, "On resampling algorithms for particle filters," in *Proc. IEEE Nonlinear Statistical Signal Processing Workshop*, 2006, pp. 79–82.

[15] D. B. Rubin, "A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The SIR algorithm," *J. Amer. Statist. Assoc.*, vol. 82, no. 398, pp. 543–546, 1987.

[16] T. Li, T. P. Sattar, and S. Sun, "Deterministic resampling: Unbiased sampling to avoid sample impoverishment in particle filters," *Signal Process.*, vol. 92, no. 7, pp. 1637–1645, 2012.

[17] J. S. Liu, R. Chen, and W. H. Wong, "Rejection control and sequential importance sampling," *J. Amer. Statist. Assoc.*, vol. 93, no. 443, pp. 1022–1031, 1998.

[18] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. Amer. Statist. Assoc.*, vol. 94, no. 446, pp. 590–591, 1999.

[19] M. Bolić, P. M. Djurić, S. Hong, "New resampling algorithms for particle filters," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2003, vol. 2, pp. 589–592.

[20] A. Budhiraja, L. Chen, and C. Lee, "A survey of numerical methods for nonlinear filtering problems," *Physica D*, vol. 230, no. 1–2, pp. 27–36, 2007.

[21] D. Crisan and T. Lyons, "A particle approximation of the solution of the Kushner-Stratonovitch equation," *Prob. Theory Relat. Fields*, vol. 115, no. 4, pp. 549–578, 1999.

[22] T. Li, T. P. Sattar, and D. Tang, "A fast resampling scheme for particle filters," in *Proc. Constantinides International Workshop on Signal Processing*, 2013, pp. 1–4.

[23] T. Li, S. Sun, and J. Duan, "Monte Carlo localization for mobile robot using adaptive particle merging and splitting technique," in *Proc. IEEE Int. Conf. Information and Automation*, 2010, pp. 1913–1918.

[24] P. Fearnhead and P. Clifford, "On-line inference for hidden Markov models via particle filters," *J. R. Statist. Soc.: Ser. B,* vol. 65, no. 4, pp. 887–899, 2003.

[25] J. Zheng, B. Bai, and X. Wang, "Increased-diversity systematic resampling in particle filtering for BLAST," *J. Syst. Eng. Electron.*, vol. 20, no. 3, pp. 493–498, 2009.

[26] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *Int. J. Robot. Res.*, vol. 22, no. 12, pp. 985–1003, 2003.

[27] T. Li, S. Sun, and T. Sattar, "Adapting sample size in particle filters through KLD-resampling," *Electron. Lett.*, vol. 46, no. 2, pp. 740–742, 2013.

[28] W. R. Gilks and C. Berzuini, "Following a moving target-Monte Carlo inference for dynamic Bayesian models," *J. R. Statist. Soc. B*, vol. 63, no. 1, pp. 127–146, 2001.

[29] C. Musso, N. Oudjane, and F. Legland, "Improving regularized particle filters," in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer, 2001, pp. 247–272.

[30] O. Hlinka, F. Hlawatsch, and P. M. Djurić, "Distributed particle filtering in agent networks: A survey, classification, and comparison," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 61–81, 2013.

[31] S. Hong, S. Chin, and P. M. Djurić, "Design and implementation of flexible resampling mechanism for high-speed parallel particle filters," *J. VLSI Signal Process.*, vol. 44, no. 1–2, pp. 47–62, 2006.

[32] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Processing*, vol. 53, no. 7, pp. 2442–2450, 2005.

[33] I. Strid, "Computational methods for Bayesian inference in macroeconomic models," Ph.D. dissertation, Stockholm School of Economics, 2010.

[34] O. Rosén, A. Medvedev, and M. Ekman, "Speedup and tracking accuracy evaluation of parallel particle filter algorithms implemented on a multicore architecture," in *Proc. IEEE Int. Conf. Control Applications*, 2010, pp. 440–445.

[35] G. Hendeby, R. Karlsson, and F. Gustafsson, "Particle filtering: The need for speed," *EURASIP J. Adv. Signal Process.*, vol. 2010, pp. 22:1–22:9, June 2010.

[36] K. Hwang and W. Sung, "Load balanced resampling for real-time particle filtering on graphics processing units," *IEEE Trans. Signal Processing*, vol. 61, no. 2, pp. 411–419, 2013.

[37] G. Hendeby, J. D. Hol, R. Karlsson, and F. Gustafsson, "A graphics processing unit implementation of the particle filter," in *Proc. European Signal Processing Conf.*, 2007, pp. 1639–1643.

[38] L. Murray, "GPU acceleration of the particle filter: The Metropolis resampler," arXiv preprint arXiv:1202.6163.

[39] A. C. Sankaranarayanan, A Srivastava, and R Chellappa, "Algorithmic and architectural optimizations for computationally efficient particle filtering," *IEEE Trans. Image Processing*, vol. 17, no. 5, pp. 737–748, 2008.

[40] B. Balasingam, M. Bolić, P. M. Djurić, and J. Míguez, "Efficient distributed resampling for particle filters," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, 2011, pp. 3772–3775.

[41] J. Míguez, M. F. Bugallo, and P. M. Djuric´, "A new class of particle filters for random dynamical systems with unknown statistics," *EURASIP J. Appl. Signal Process.*, vol. 15, pp. 2278–2294, Jan. 2004.

[42] T. S. John, A. Nallanathan, and M. A. Armand, "A non-resampling sequential Monte Carlo detector for coded OFDM systems based on periodic termination of differential phase trellis," *IEEE Trans. Wireless Commun.*, vol. 5, no. 7, pp. 1846–1856, 2006.

[43] J. Kotecha, and P. M. Djurić, "Gaussian particle filtering," *IEEE Trans. Signal Processing*, vol. 51, no. 10, pp. 2592–2601, 2003.

[44] T. Chen, T. B. Schön, H. Ohlsson, and L. Ljung, "Decentralized particle filter with arbitrary state decomposition," *IEEE Trans. Signal Processing*, vol. 59, no. 2, pp. 465–478, 2011.

[45] J. Míguez, "Analysis of parallelizable resampling algorithms for particle filtering," *Signal Process.*, vol. 87, no. 12, pp. 3155–3174, 2007.

[46] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *J. Amer. Statist. Assoc.*, vol. 9, no. 425, pp. 278–288, 1994.

[47] N. Celik and Y.-J. Son, "State estimation of a shop floor using improved resampling rules for particle filtering," *Int. J. Prod. Econ.*, vol. 134, no. 1, pp. 224–237, 2011.

[48] A. A. Nasir, S. Durrani, and R. A. Kennedy, "Particle filters for joint timing and carrier estimation: Improved resampling guidelines and weighted Bayesian Cramér-Rao bounds," *IEEE Trans. Commun.*, vol. 60, no. 5, pp. 1407–1419, May 2012.

[49] R. Douc and E. Moulines, "Limit theorems for weighted samples with applications to sequential Monte Carlo," *Ann. Stat.*, vol. 36, no. 5, pp. 2344–2376, 2008.

[50] P. Fearnhead, "Computational methods for complex stochastic systems: A review of some alternatives to MCMC," *Stat. Comput.*, vol. 18, no. 2, pp. 151–171, 2008.

[SP]