NEW RESAMPLING ALGORITHMS FOR PARTICLE FILTERS

Miodrag Bolić, Petar M. Djurić, and Sangjin Hong

Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY, 11794-2350
mbolic, djuric, snjhong@ece.sunysb.edu

ABSTRACT

Resampling is a critically important operation in the implementation of particle filtering. In parallel hardware implementations, resampling becomes a bottleneck due to its sequential nature and the increased complexity it imposes on the traffic of the designed interconnection network. To circumvent some of these difficulties, we propose two new resampling algorithms. The first one, called residual-systematic resampling, combines the merits of both systematic and residual resampling and is suitable for pipelined implementation. It also guarantees fixed duration of the resampling procedure irrespective of the weight distribution of the particles. The second algorithm, referred to as partial resampling, has low complexity and reduces traffic load through the hardware network. These two algorithms should also be considered as resampling methods in simulations on standard computers.

1. INTRODUCTION

Particle filters perform three basic operations sequentially: generation of new particles (sampling step), computation of particle weights (importance step), and resampling [5]. The resampling step is critical in every implementation of particle filtering because without it, the weights of the particles quickly become so diverse that inference is made by using only a very small number of particles. The idea of resampling is to remove particle trajectories with small weights and replace them with trajectories with large weights. Resampling is an important statistical tool [11], and it was proposed for use in particle filtering in various works including [2, 8, 9]. A detailed theoretical discussion of resampling in the particle filtering context is given in [9].

Standard algorithms used for resampling are different variants of stratified sampling such as residual resampling (RR) [1], branching corrections [4], systematic resampling (SR) [5, 6] as well as resampling methods with rejection control [10]. In this paper, resampling algorithms are approached mainly from the standpoint of parallel hardware implementation. In contrast to the sampling and importance steps which are suitable for parallel implementation, resampling introduces full data dependencies that make the parallelization of the resampling step very difficult. In addition, this step produces surviving particles which undergo further processing and which in a parallel scheme require intense communication among the hardware elements.

In this paper we propose two new resampling algorithms, which are developed by considering hardware implementation of the particle filters. The main feature of the first resampling algorithm, referred to as residual-systematic resampling (RSR) and described in Section 3, is to perform resampling in fixed time even if the number of particles at the input and output of the resampling procedure is not the same. One such application is in tracking with particle filter having variable number of particles [3]. Using this algorithm, maximum speed can be achieved where the resampling is parallelized in the sense that the particles are partitioned into groups and resampled concurrently. In addition, RSR simplifies the control in parallel implementations, while producing identical results as the SR algorithm. In sequential applications (of MATLAB-type, for example) we found that it is faster than the SR algorithm. When the number of particles is the power of two, it becomes simpler and even faster.

The second algorithm, called partial resampling (PR) (Section 4), addresses mainly communication issues. In the PR algorithm, resampling is carried out only for particles with significant weights, thereby reducing the resampling and communication times.

2. BRIEF REVIEW OF THE SYSTEMATIC RESAMPLING AND RESIDUAL RESAMPLING ALGORITHMS

In this section, resampling algorithms are reviewed with worst case analysis for determination of the sampling rate in hardware implementations. Note that here we do not consider rejection control algorithms because they are not suitable for high speed implementations. Their time for resampling cannot be determined beforehand because the execution time itself is a random variable. In other words, the algorithm requires random number generation, where the number of random draws cannot be predicted and is variable due to random rejections.

The SR algorithm performs resampling in the same way as the basic random resampling algorithm, with one exception. Instead of drawing each U_m independently from $\mathcal{U}(0,1)$ for $m = 1, ..., M$, where M is the number of resampled particles, it uses a uniform random number U according to $U \sim \mathcal{U}[0, \frac{1}{M}]$, and $U_m = U + (m-1)/M$. One possible algorithm for systematic resampling is as follows:

(i)=SR(N, M)
Generate random number $U \sim \mathcal{U}[0, \frac{1}{M}]$
$s = 0$
for $m = 1 : N$
$k = 0$
$s = s + w(m)$

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

```
    while (s > U)
      k = k + 1
      U = U + 1/M
    end
    i(m) = k
end
```

Pseudocode 1: Systematic resampling (SR) algorithm.

In Pseudocode 1, N is the input number of particles, M is the number of particles generated after resampling, and w is an array of scaled weights from the importance step. The output i is an array of indexes, which shows how many times each particle is replicated. We observe that SR is implemented using *two loops*.

The RR algorithm with a simple modification of the one presented by [1] can be implemented as follows:

```
(i) = RR(N, M)
M_r = M
for m = 1 : N
    i(m) = ⌊w(m) · M⌋
    w(m) = w(m) · M − i(m)
    M_r = M_i − i(m)
end
if M_r > 0
    for m = 1 : N
        w(m) = w(m)/M_r
    end
    (i_r) = SR(N, M_r)
    for m = 1 : N
        i(m) = i(m) + i_r(m)
    end
end
```

Pseudocode 2: Residual resampling (RR) algorithm.

Here N and M have the same meaning as before, and w is an array of scaled weights from the importance step. The output i, as before, is an array of indexes, which shows how many times each particle is replicated. We should note that RR is composed of two steps. In the first step, the number of replications of particles is calculated. Since this method does not guarantee that the number of resampled particles is M, the residual M_r is computed. The second step requires resampling which produces M_r of the final M particles. In Pseudocode 2, this step is performed by SR.

The best case in terms of speed of execution of the RR algorithm occurs when $M_r = 0$ (for example, when ${w_t}^{(m)}M$ is an integer for all $m = 1, 2, \cdots M$). In that case there is only one step with m iterations. In other situations, there are two steps, where the first one is the same as before, and the second step amounts to resampling of the residuals. The worst case of RR arises when $M_r = N-1$. One example of that case is, when one particle has a weight in the range $[1/N, 2/N)$ and the remaining $N-1$ particles have weights less than $1/N$. Then, step 2 requires generation of $N-1$ random numbers.

3. THE RESIDUAL-SYSTEMATIC RESAMPLING ALGORITHM

The newly proposed resampling algorithm is based on stratified resampling and the ideas behind the RR and SR algorithms. Hence we refer to it as residual systematic resampling (RSR). Similarly to RR, RSR calculates the number of times each particle is replicated except that second iteration of the residual resampling by using a special way of drawing random numbers for systematic resampling.

In RR, the number of replications of a specific particle is determined by truncating the product of the number of particles and the particle weight. In RSR, instead of using only weights in the product, weights are subtracted from the updated uniform random number formed in similar fashion as in SR. The algorithm has only *one loop* and the processing time is independent of the input data. The RSR algorithm is summarized by the following pseudocode:

```
(i) = RSR(N, M)
Generate a random number U ~ U [0, 1/M]
for m = 1 : N
    i(m) = ⌊(w(m) − U) · M⌋ + 1
    U = U + i(m)/M − w(m)
end
```

Pseudocode 3: Residual systematic resampling (RSR) algorithm.

The resampling result obtained using RSR is identical to the one using systematic resampling. In Figure 1, we present graphically the two methods, where the difference between them can be captured. It is shown that for the same uniform number U, the positions of U after updating are the same for the two methods. The only difference is that in systematic resampling, U is calculated with reference to the origin of the cumulative sum of weights, while in RSR resampling, the number U is updated with reference to the origin of the currently considered weight. For this reason, the value of the weight of the previous particle must be subtracted from U.

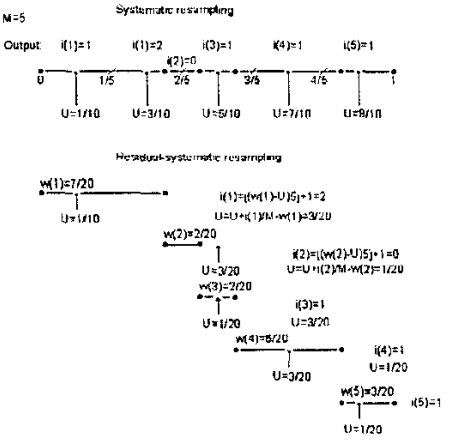

Fig. 1. The systematic and residual-systematic resampling methods for an example in which the number of particles is $M = 5$.

Even though RSR yields the same result as SR, the former has the following advantages:

1. The RSR algorithm contains only one loop.
2. The complexity of the algorithm is always $O(N)$ even if the number of particles after resampling is greater than the initial number of particles.

3. RSR is suitable for pipelining because it can be implemented without conditional branches.

4. PARTIAL RESAMPLING

The idea of partial resampling is to perform resampling only on particles with large weights and replace them with particles with negligible weights. Particles with moderate weights are not resampled. The advantages of this method are the following:

1. resampling is done faster because it is done on a much smaller number of particles, and
2. communication is shorter since less particles are replicated and replaced.

The main disadvantage is that after resampling the weights are not equal and that for the calculation of new weights in the importance step at time t, the weight values at time $t-1$ are used.

4.1. Partial stratified resampling

In partial stratified resampling (PSR), resampling is preceded by a step that groups the particles according to their weights in three sets. The weight of each particle is compared with a high and a low thresholds, T_h and T_l, respectively. Particles with weights between those two thresholds are considered moderate and are not resampled. 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 weights of the particles that are resampled is computed using $S_{hl} = \sum_{j=1}^{N_h+N_l} w_t{}^{(j)}$, where j is chosen so that the conditions $w_t^{(j)} > T_h$ or $w_t^{(j)} < T_l$ are satisfied. Then, SR or RSR is done only on particles whose weights satisfy $w_t^{(j)} > T_h$ or $w_t^{(j)} < T_l$.

The PSR consists of two loops. The first one contains N iterations and is used for classifying the particles as dominant, moderate, or negligible. The second loop has $(N_l + N_h)$ iterations, which equals the number of particles involved in the resampling. In the end, a new random measure is produced, given by $\{\widetilde{\mathbf{x}}_{1:t}^{(m)}, \widetilde{w}_t^{(m)}\}_{m=1}^{M}$ where:

$$\widetilde{w}_t^{(j)} = \begin{cases} 1/S_{hl}, & \text{for } w_t^{(j)} > T_h \text{ or } w_t^{(j)} < T_l, \\ w_{t-1}^{(j)}, & \text{otherwise} \end{cases}.$$

The PSR algorithm, where RSR is applied for resampling, requires passing through two loops of N iterations. We can also show that this method provides unbiased estimates. With PSR, the communication requirements for the worst case are the same as for SR, RR, or RSR. This case occurs when $N_l + N_h = N$, which means that all the particle must be resampled. This implies, that there cannot be improvements from an implementation standpoint. It should be noted that there are several ways of speeding up the PSR algorithm and improving the communication during updating of the states. One of them is described in the next subsection.

4.2. Partial deterministic resampling

For partial deterministic resampling (PDR), we again define two thresholds T_h and T_l, but now all the particles with weights less than T_l are removed, the ones with weights between T_h and T_l are unchanged, and the particles with weights larger than T_h are replicated. Each particle of N_h is replicated n_i number of times, where n_i is the number equal (with error of one particle) for all N_h particles.

For the purpose of presentation, we will assume that the particles are ordered as follows: dominating particles, negligible and moderate particles. When $N_h, N_l > 0$, the weights of the particles, $\widetilde{w}_t^{(i)}$, and n_i are computed by

$$n_i = \begin{cases} \lfloor \frac{N_t}{N_h} \rfloor + 2, & 0 < i \le N_t \\ \lfloor \frac{N_t}{N_h} \rfloor + 1, & N_t < i \le N_h \end{cases}$$

$$\widetilde{w}_t^{(i)} = \begin{cases} \frac{w_t{}^{(i)}}{n_i}, & 0 < i \le N_h \\ \frac{w_t{}^{(j)}}{n_{\lfloor \frac{i}{N_h} \rfloor}} + w_t{}^{(\lfloor \frac{i}{N_h} \rfloor)}, & N_h < i \le N_h + N_l \\ w_{t-1}{}^{(i)}, & N_h + N_l < i \le N, \end{cases}$$

where, $N_t = \lfloor \frac{N_l}{N_h} \rfloor N_h$. If N_h and/or N_l are equal to zero, resampling is not carried out.

We tested the performance of this method by applying it to the bearings-only tracking problem with different initial conditions. For this type of experiment, three sets of threshold values were used, i.e., $T_h = \{2M, 5M, 10M\}$ and $T_l = \{1/(2M), 1/(5M), 1/(10M)\}$. In Figure 2, for the three different pairs of thresholds we show the number of times when the track is lost versus number of particles. The used algorithms are SR, SR performed every 5-th observation, and PDR. Another set of experiments was performed for fixed T_h and variable T_l and vice versa. The results show that the number of times when tracking is lost decreases when T_l decreases for fixed T_h. The reason for this can be sought in the fact that some particles with weights bellow T_l could also carry useful information especially in the case when the observation noise has large variance. In the case of large T_l (such as $1/(2M)$) those valuable particles would be removed. For fixed T_l and as we increase T_h, the number of times when the track is lost also increases. The reason for losing tracks more frequently than for the case when resampling is performed every time is that, again, particles with small weights which are very important in the case of jumps are removed if the value of T_h is high.

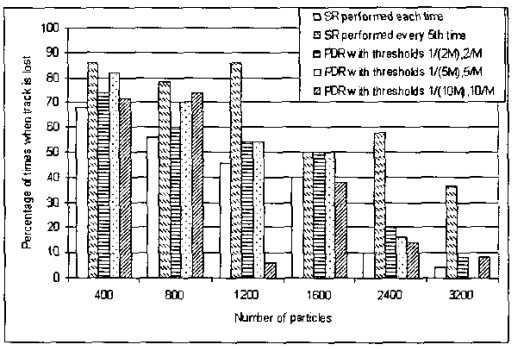

Fig. 2. Bearings-only tracking example: number of times when track is lost for different the number of particles M and different threshold values

5. DISCUSSION

We reiterate that the complexity of the RR and RSR and PDR algorithms is of $O(N)$, and that of the SR algorithm, of $O(\max(N, M))$. In Table 5 we provide a comparison of the different resampling

algorithms in terms of number of operations for a MATLAB-type implementation. The results for RR are obtained for the worst case scenario.

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

Table 1. Comparison of the number of operations for different resampling algorithms.

When implementing the resampling algorithms, conditional branches degrade the performance of an instruction pipeline. Due to the random nature of the weights, techniques such as branch-prediction buffer or branch history are not useful for branching prediction in the SR case. So, the "while" loop in the resampling step is undesirable because it makes the implementation of efficient pipelining difficult. The RSR algorithm can be put in a form very suitable for high-speed implementations.

In Table 1, the time for exchanging particles after the update step was not taken into account. The number of particles that should be exchanged is the same for all stratified resampling algorithms. However, in the case of PDR, the maximum number of exchanging particles is N/T_h, and that could represent a significant saving in the amount of traffic through the interconnection network in a parallel implementation. On the other hand, PDR makes the subsequent importance step more complicated since additional operations are necessary during calculation of the weights.

In Figure 3, we present comparison results of the times of execution of the SR, RSR, and PR algorithms. Three experiments were performed on a standard PC, where in each experiment the number of particles was $N = M = 1000, 2000$, and 4000. For the PR algorithm, we used $T_h = \{2/M, 5/M, 10/M\}$ and $T_l = \{1/(2M),\ 1/(5M),\ 1/(10M)\}$. The results show that, as expected, the PR algorithms were the fastest (of which the one with $T_h = 10/M$ and $T_l = 1/(10M)$ was the quickest, since then the smallest number of particles was resampled.) It has to be kept in mind that the overall performance of particle filters that use PR is usually poorer than the performance of particle filters that employ SR or the RSR method. It is important to notice the difference between the SR and RSR, where RSR outpaces SR by 15%.

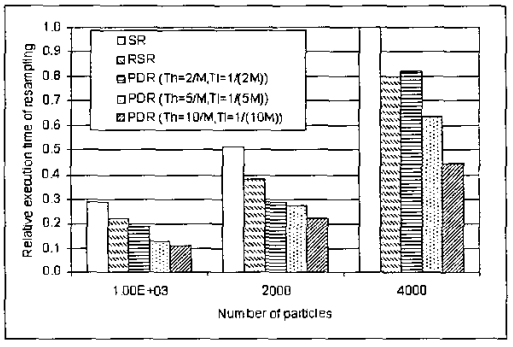

Fig. 3. Relative duration of different resampling algorithms executed on a standard PC.

6. CONCLUSION

In this paper two new resampling methods are proposed. RSR is based on stratified resampling, and it produces identical resampling results as SR. However, RSR is faster and its duration of resampling is the same for the fixed number of input particles regardless of the output number of particles. PR is a threshold based method in which only particles with weights above the threshold are resampled. Comparisons of the new algorithms with SR show that new algorithms have only one loop (instead of two as in SR). The communication requirements of PR are the lowest, but its performances is worse than that of SR or RSR. In addition, it produces non-equal weights after resampling which leads to accumulation of error in calculating weights when finite precision arithmetics is applied.

7. REFERENCES

[1] E. R. Beadle and P. M. Djurić, "A fast weighted Bayesian bootstrap filter for nonlinear model state estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, pp. 338-343, 1997.

[2] 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, pp. 1403-1412, 1997.

[3] M. Bolić, S. Hong and P. M. Djurić, "Performance and Complexity Analysis of Adaptive Particle Filtering for Tracking Applications", to appear in Proceedings of the 36th IEEE Asilomar Conference on Signals, Systems, and Computers (Asilomar'2002), Pacific Grove, CA, November 2002.

[4] D. Crisan, P. Del Moral, and T. J. Lyons, "Non-linear filtering using branching and interacting particle systems," *Markov processes and Related Fields*, vol. 5, no. 3, pp. 293-319, 1999.

[5] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*, New York: Springer Verlag, 2001.

[6] N. J. Gordon, D. J. Salmond, and A. F. M. Smith,"A novel approach to nonlinear and non-Gaussian Bayesian state estimation," IEE Proceedings F, vol. 140, pp. 107-113, 1993.

[7] N. J. Gordon, D. J. Salmond, and C. Ewing, "Bayesian state estimation for tracking and guidance using the bootstrap filter," Journal of Guidance, Control and Dynamics , vol. 18, pp. 1434-1443, 1995.

[8] 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.

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

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

[11] D. B. Rubin, J. M. Bernardo, M. H. De Groot, D. V. Lindley, and A. F. M. Smith, *Bayesian Statistics 3*, Oxford: University Press, pp. 395-402, 1988.