# Finite Precision Effect on Performance and Complexity of Particle Filters for Bearing-Only Tracking*

Miodrag Bolić, Sangjin Hong, and Petar M. Djurić
Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11794-2350
{mbolic, snjhong, djuric}@ece.sunysb.edu

## Abstract

*In this paper complexity reduction of particle filters and their performance due to finite precision effects are investigated. A method for dynamic range scaling of importance weights is proposed as well as ways for its integration with computation of the weights. Functions are approximated in a way to mitigate dynamic range scaling problems. To reduce complexity, redundant operations of particle filters are identified and particle filters are correspondingly modified. We consider the sample importance resampling (SIR) method of particle filtering applied to the bearings-only tracking problem.*

## 1 Introduction

In the recent past, particle filters have attracted much attention as a methodology for tracking and/or detection of dynamic signals [1]. Particle filters outperform other filters, such as the Kalman filter, in many important practical situations, and their flexibility in addressing a wide variety of problems makes them very appealing. However, the hardware complexity of the particle filtering can be significant for practical VLSI implementation. One way to reduce hardware complexity is by employing finite precision processing by using fixed point arithmetics [4, 3] and complex function approximations [5].

The size of the computation metrics such as word length as well as the number and type of operations are commonly used indicators of performance in signal processing applications. Intuitively, the larger the computation, the larger the area, power and delay one should expect. In particle filters, the computation's size is proportional to the number of particles $M$. The main computational burden of particle filters is related to the importance step, where new weights are calculated. It requires $M$ calculations of complex mathematical functions (exponential, arctan, division, square) for each input sample. The use of finite word size for representing variables affects the dynamic range of data representation, which is particularly apparent when exponential functions are computed. In a case of large deviations of the observation, the weights are dispersed and many of them have negligible values. In this case, tracking would be lost if the dynamic range of the exponential function is not extended.

In this paper, a method for scaling the dynamic range has been proposed. The main idea is to cover a wide dynamic range with small number of bits for representing exponential functions (16 or less). The method operates on the exponents of the exponential function used for the weight calculation.

The reduction of number of operations is a very important task. It has been shown that particle filters applied to the bearings-only tracking problem [2], which initially perform $2M$ divisions per each observation, can be implemented with only one division. The exclusion of division helps not only in reducing the number of operations but in avoiding dynamic range problems due to division.

## 2 Algorithmic representation of particle filter steps

In order to provide a better illustration of the order of computational complexity of particle filters, we present a pseudo-code of the particle filter for the bearings-only tracking problem. Particle filters sequentially update its random measure $(X, W)$ based on new observations and the random measures from the previous time instants. The symbol $X$ is the hidden state which consists of the position and velocity of the tracked object in the Cartesian coordinate system $(X = \{x, V_x, y, V_y\})$ and $W$ represents the importance

*This work has been supported under Awards CCR-9903120 and CCR-0220011.

0-7803-7576-9/02$17.00 © 2002 IEEE

weights associated with $X$. In Pseudocode 1, a particle filter code for processing one observation is presented. An input argument to the particle filter is the observation $z$, and the output are the estimates of the system states $\hat{X} = \{\hat{x}, \hat{V}_x, \hat{y}, \hat{V}_y\}$. The symbols in the pseudocode, $n_x, n_y$, denote two $M$-dimensional arrays of Gaussian random numbers used in the sample step. This routine should be called from an infinite loop. It performs sequentially the following three steps: sampling, computation of the importance weights, and resampling. There are two additional steps, one of which is used for updating the states after resampling, and the other, used for calculating the estimates. In this algorithm, resampling is performed at each time instant. The algorithm requires adding a criterion for resampling when resampling is not performed at each time instant.

$(\hat{x}, \hat{V}_x, \hat{y}, \hat{V}_y) = PF(z, n_x, n_y)$
  $(x, V_x, y, V_y) = BOT_S(\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}, n_x, n_y)$
  $(w, S_M) = BOT_I(x, y, \widetilde{w}, z)$
  $(i) = SR(1, w)$
  $(\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}, \widetilde{w}) = BOT_U(x, V_x, y, V_y, i)$
  $(\hat{x}, \hat{V}_x, \hat{y}, \hat{V}_y) = BOT_O(x, V_x, y, V_y, w)$

Pseudocode 1: Processing of one observation by a particle filter.

The generation of particles is performed by drawing them from the importance density in the sampling step. In Pseudocode 2, the sampling step for the bearings-only tracking is presented. The input arguments are the states of the particles obtained from the update state step from the previous time instant ($\widetilde{X} = \{\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}\}$). For sake of simplicity of implementation, the prior density of the state is selected as importance density. The output of the sampling step is a new vector of states $X = \{x, V_x, y, V_y\}$.

$(x, V_x, y, V_y) = BOT_S(\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}, n_x, n_y)$
  for m=1:M
    $x(m) = \widetilde{x}(m) + \widetilde{V}_x(m) + 0.5n_x(m)$
    $V_x(m) = \widetilde{V}_x(m) + n_x(m)$
    $y(m) = \widetilde{y}(m) + \widetilde{V}_y(m) + 0.5n_y(m)$
    $V_y(m) = \widetilde{V}_y(m) + n_y(m)$
  end

Pseudocode 2: Sampling step.

One possible realization of the importance step which is used for weight calculations is shown in Pseudocode 3. The importance step consists of two substeps: weight calculation and normalization. First, the weights are evaluated up to a proportionality constant and subsequently, they are normalized. The input arguments are the observation $z$, the arrays of states $x$ and $y$ and updated weights from the previous time instant $\widetilde{w}$.

$(w, S_M) = BOT_I(x, y, \widetilde{w}, z)$
  *//Calculation of weights and their sum*
  $S_M = 0$
  for $m = 1 : M$
    $w^*(m) = \widetilde{w}(m) e^{-(2\pi\sigma_v^2)^{-1} \cdot (z - atan\frac{y(m)}{x(m)})^2}$
    $S_M = S_M + w^*(m)$
  end
  *//Normalization*
  for $m = 1 : M$
    $w(m) = \frac{w^*(m)}{S_M}$
  end

Pseudocode 3: Importance step.

In Pseudocode 4, a systematic resampling algorithm is presented. There, $w$ is an array of scaled weights from the importance step. When the weights are normalized, the input sum of weights $S_M$ is 1. The output $i$ is an array of indexes, which shows the new positions of the resampled particles.

$(i) = SR(S_M, w)$
  $A_d = \frac{S_M}{M}$
  Generate random number $U \sim U[0, A_d]$
  $S = 0, k = 0$
  for $m = 1 : M$
    while $(S < U)$
      $k = k + 1$
      $S = S + w(k)$
    end
    $U = U + A_d$
    $i(m) = k$
  end

Pseudocode 4: A systematic resampling algorithm.

The particles have to be updated (Pseudocode 5) in order defined by the index array $i$ from the resampling step. The output of the updated states is the new random measure $(\widetilde{X}, \widetilde{W})$, where $\widetilde{X} = \{\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}\}$ and $\widetilde{w}(m) = 1/M$ for $m = 1, ..., M$. The states updating is performed simply by indirect addressing $x(i(m))$ using the index array $i(m)$.

$(\widetilde{x}, \widetilde{V_x}, \widetilde{y}, \widetilde{V_y}, \widetilde{w}) = BOT_U(x, V_x, y, V_y, i)$
  for m=1:M
    $\widetilde{x}(m) = x(i(m))$
    $\widetilde{V}_x(m) = V_x(i(m))$
    $\widetilde{y}(m) = y(i(m))$
    $\widetilde{V}_y(m) = V_y(i(m))$
    $\widetilde{w}(m) = 1/M$
  end

Pseudocode 5: Updating of states.

The computation of the output estimates is shown in Pseudocode 6.

$(\hat{x}, \hat{V}_x, \hat{y}, \hat{V}_y) = BOT_O(x, V_x, y, V_y, w)$
  $\hat{x} = \sum_{m=1}^{M} w(m)x(m)$
  $\hat{V}_x = \sum_{m=1}^{M} w(m)V_x(m)$
  $\hat{y} = \sum_{m=1}^{M} w(m)y(m)$
  $\hat{V}_y = \sum_{m=1}^{M} w(m)V_y(m)$

Pseudocode 6: Computation of estimates.

## 3 Computational kernel

An important property of signal processing algorithms is that their execution time and energy are

dominated by regular, repetitive kernels of computation. These kernels are calculations that are performed in the inner loops of a program implementing a given DSP algorithm. The performance of particle filters for the bearings-only tracking problem are evaluated on the popular TI TMS320C54x DSP processor [6]. The execution times of the particle filter operations based on the algorithms from Section 2 are shown in Table 1. Here, $S$ represents sampling, $I$ the computation of the importance weights, $R$ resampling, and $O$ the computation of the output estimates.

Table 1: Execution times of particle filter operations for the TMS320C54x processor.

| PF operation | S | I | R | O |
|---|---|---|---|---|
| Duration (%) | 12 | 67 | 11 | 10 |

The computation of the importance weights takes most of the time (about 67%) as all the non-linear computations take place there. The type of non-linear functions depends on the problem at hand, but the most common non-linear function is the exponential function because a standard assumption used in modelling is that the observation noise is Gaussian. We must say here that the execution time for sampling is calculated without including time for Gaussian random number generation because the random number generation is not in the critical path.

# 4 Reducing the number of operations

The number of operations is reduced by recognizing redundant operations and by replacing operations with simpler ones whenever possible.

## 4.1 Integrating normalization of weights with other steps

The normalization step (Pseudocode 3) requires the use of an additional loop of $M$ iterations as well as $M$ divisions per observation. It has been noted that the normalization represents an unnecessary step which can be merged with the resampling and the computation of the importance weights. Necessary changes in Pseudocode 3 include no normalization step and no normalization of the weights at the output. Avoiding normalization requires additional changes which depend on whether resampling is carried out at each time instant. For particle filters which perform resampling at each time instant, the following arguments should be used when the resampling routine is called from Pseudocode 1: $(i) = SR(S_M, w^*)$. Since the weights are not normalized, the uniform random number in the systematic resampling routine should be drawn from $[0, S_M/M)$ and updated with $S_M/M$.

When particle filters do not perform resampling at each time instant, modifications in the computation of the importance weights are necessary. In Pseudocode 3 we can see that the calculation of new weights requires multiplication with the weight from the previous time instant. When the weights are not scaled, this multiplication could cause dynamic range problems, and therefore, the weight should be scaled at this point. However, in order to avoid division with the sum of the weights $S_M$, $\ln(S_M)$ is calculated and added to the exponent of the exponential function.

Using this approach, only one division is performed in the resampling step ($S_M/M$), and that significantly reduces the dynamic range problem for fixed precision arithmetics which usually appears with division. The computational burden is reduced as well since the normalization requires $M$ divisions.

## 4.2 Reducing the number of multiplications

There are several places in the filter where multiplication is unnecessary or it can be replaced with simpler operations. $M$ multiplications in the computation of the importance weights can be replaced with additions. Instead of multiplying the exponential function with the weight from the previous time instant, one can use only the exponent of that weight, and thereby convert multiplication to addition.

Multiplications can be avoided in the calculation of the output estimates for particle filters that perform resampling at each time instant. The state estimates can be computed from the new random measure $(\widetilde{X}, \widetilde{w})$, for which the multiply-accumulate operation in Pseudocode 6 is replaced with the accumulate operation because all the weights are equal.

After applying all the modifications for reducing computing time, our comparisons made on TMS320C54x showed that the execution time was reduced about 20%. The comparison was made on an algorithm that performs resampling at each time instant.

# 5 Finite Precision Processing

Processing with finite precision has been a problem for many signal processing applications requiring minimal amount of hardware. Most of the design efforts have been in reducing the hardware resources while maintaining the performance level. In this section, we examine two aspects of finite precision processing: the finite word length effect and approximate processing.

One of the most important qualities of particle filters is that the error of the weight calculation due to finite precision effects does not accumulate during

time. It is a result of resampling after which all particles have equal weights. This means that after resampling, the "computational history" of the weight is erased. Accumulation of errors, however, exists in the sampling step since the states from the previous time instant are used for proposing new particles. There, too, resampling reduces error accumulation by replacing particles whose weights are low due to the statistical nature of tracking and/or computational errors. So resampling reduces the ill-effects of finite precision processing.

Next, finite precision effects on the particle filter operations from Section 2 are considered. The sampling step consists of additions which do not give rise to real finite precision problems. Approximation of the Gaussian random number generators with generators suitable for parallel hardware implementation has not been studied here. Finite word length effects have been considered, and it has been shown that satisfying accuracy of tracking can be achieved when all the variables in the sample step are represented with 14 bits.

The computation of the importance weights will be considered separately in Section 5.1 since almost all finite precision problems of particle filters originate from this operation.

Operations that are performed in resampling are simple additions. However, the resampling is very sensitive to any kind of truncation or rounding during additions because it can give rise to non-constant number of particles at the output of the resampling unit. So, adders used for resampling must be implemented with higher number of bits in order to avoid this error.

The update step does not contain any arithmetic operations, and therefore it is not affected by finite precision.

### 5.1 Effect of Finite Word Length in the Computation of Importance Weights

The main computational burden of particle filters is related to the computation of the importance weights. In the case of SIR algorithm used for tracking, the importance step requires calculation of the following: exponentiation, finding the arctan, squaring, multiplications, and divisions. The representation of the outputs of all these functions in fixed point arithmetic causes errors due to cumulative finite word length effects.

The dynamic range problems are critical for the exponential functions. In order to understand these problems, the process of weight propagation is studied. In situations when the particle filter tracks well, the particles are drawn from a region around the actual state values, and consequently, the non-normalized weights are large. On the other hand, when there are significant deviations from the most recent estimate, the values of the weights can be very small. Then, tracking may be lost if the dynamic range is not scaled.

A method for scaling the dynamic range has been proposed in Pseudocode 7. It also provides a procedure for merging the scaling with the importance step. The main idea is to cover wide dynamic ranges with low number of bits for representing exponential functions (16 or less). The scaling method operates on the exponents of the exponential function used for calculating the weights. It consists in finding the minimum of the exponents of all the particles and then scaling all the exponents with this minimum. The method uses the property of particle filters that it is sufficient to know the weights of the particles up to a proportionality constant. The dynamic scaling of the range then is not acute since the maximum value of the weights after computation of the exponential function is one.

The scaling method requires two loops of $M$ iterations in the weight calculation step as is shown in Pseudocode 7. One loop is used for calculating the exponents of the exponential functions and finding the minimum exponent, while the other serves for calculation of the weights and computation of their sum. The disadvantage of this method is that the extra loop increases the execution time.

$(w, S_M) = BOT_I(x, y, \tilde{w}, z)$
  *//Calculation of the exponents and their minimum*
  $S_M = 0, min = MAX$
  for $m = 1 : M$
    $a(m) = (2\pi\sigma_v^2)^{-1} \cdot (z - atan\frac{y(m)}{x(m)})^2$
    if $(a(m) < min)$ $min = a(m)$;
  end
  *//Calculation of the weights and their sum*
  for $m = 1 : M$
    $w^*(m) = \tilde{w}(m)e^{-a(m)+min}$
    $S_M = S_M + w^*(m)$
  end
  *//Normalization*
  for $m = 1 : M$
    $w(m) = \frac{w^*(m)}{S_M}$
  end

Pseudocode 7: The computation of importance weights with a scaling method.

When the dynamic range is known, the computation of the exponential function could be performed only for particles that can contribute to the generation of new particles. In that way, the number of calculations of the exponential function is reduced which is especially obvious when the filter does not track well. This property allows for adding additional logic to improve tracking in critical situations such as jumps in the input signal. Simulations show that if the input

signal changes its value between two consecutive samples for more than 20%, there are more than 70% of particles that are too small to contribute to the output.

Aside from the exponential function and the division in the *atan* function, the ranges of all the other functions in the importance step are known in advance, so that a dynamic change of the fixed point is not necessary. The simulation results show that even 12 bits of resolution for the exponential function provide good accuracy at the output.

### 5.2 Function approximation

The COordinate Rotation DIgital Computer (CORDIC) shows very good properties for calculation of hyperbolic functions [5, 7]. The same CORDIC core can be used for calculation of both exponential and *atan* functions, and that will reduce area requirements. Secondly, it approximates the $atan(y/x)$ function without calculating division. In this way, dynamic range problems due to division in the *atan* function are resolved. In our simulations 14 bits of resolution gave good results in approximating those functions.

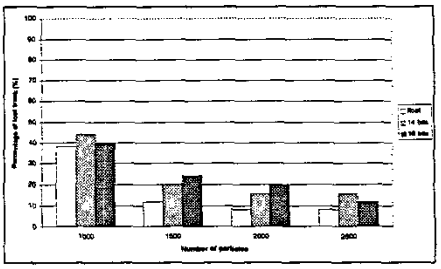
### 5.3 Simulation results

Some simulation results are displayed in Figure 1. Graph (a) shows the number of times when the track is lost versus the number of particles for 14 and 16 bit precisions and for double precision arithmetic. We consider that the track is lost if the sum of all non-normalized and non-scaled weights is very close to zero. Bellow 14 bits of resolution, the percentage of lost track increases significantly, so that 14 is the minimum number of bits that should be used for function representation. As expected, the number of times when the track is lost is higher when finite precision arithmetic is applied, and it is inversely proportional to the number of bits. Graph (b) presents the average of the mean square error (MSE) versus time for 14 and 16 bit precisions and for double precision arithmetic. The results suggest that when enough bits for finite precision are used, there is no deterioration in the accuracy of tracking. From the results, it seems that with finite precision arithmetic more extreme results were obtained: the filter either lost the track or tracked very well.

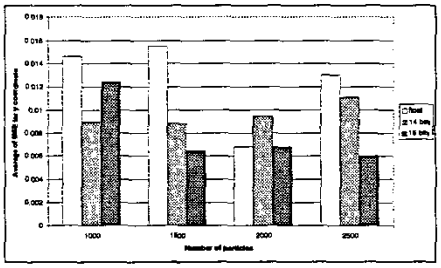
The finite precision effects of particle filters were evaluated using SystemC language for hardware-software co-design. The scaling method was used for dynamic range problems of the exponential function.

## 6 Conclusions

In this paper, we investigated complexity reduction of particle filters applied to the bearings-only tracking problem. We considered reducing the number of operations and applying finite precision processing as ways for reducing the hardware complexity. Functions that can cause dynamic range problems were identified. It was shown that particle filters can be implemented without divisions. To present exponential functions with low number of bits, a new scaling method was developed. Approximation of hyperbolic functions by using CORDIC core was described.



(a)



(b)

Figure 1: (a) Percentage of cases when tracking is lost versus the number of particles. (b) MSE versus time with 14 and 16 bits precisions and with double precision.

## References


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

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

[3] R. M. Gray and D. L. Neuhoff, "Quantization," IEEE Transactions on Information Theory, October 1998.

[4] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, McGraw-Hill, 2nd edition, 2001.

[5] J. M. Muller, *Elementary Functions, Algorithms and Implementation*, Birkhauser Boston, May, 1997.

[6] Texas Instruments, TMS320C54x DSP Library Programmers Reference, August 2002.

[7] J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Trans. Electron. Comput., vol. EC-8, pp. 330-334, 1959.