Particle Filter Acceleration Using Multiscale Sampling Methods

Yaniv Shmueli
School of Computer Science
Tel Aviv University
yaniv.shmueli@cs.tau.ac.il

Gil Shabat
School of Electrical Engineering
Tel Aviv University
gil@eng.tau.ac.il

Amit Bermanis
School of Mathematics
Tel Aviv University
amitberm@post.tau.ac.il

Amir Averbuch
School of Computer Science
Tel Aviv University
amir@math.tau.ac.il

***Abstract*—We present a multiscale based method that accelerates the computation of particle filters. Particle filter is a powerful method that tracks the state of a target based on non-linear observations. Unlike the conventional way that calculates weights over all particles in each cycle of the algorithm, we sample a small subset from the source particles using matrix decomposition methods. Then, we apply a function extension algorithm that uses the particle subset to recover the density function for all the rest of the particles. As often happens, the computational effort is substantial especially when tracking multiple objects takes place. The proposed algorithm reduces significantly the computational load. We demonstrate our method on both simulated and on real data such as tracking in videos sequences.**

***Index Terms*—particle filter, multiscale methods, nonlinear tracking**

I. Introduction

Particle filter (PF) is a powerful method for target state tracking based on non-linear observations obtained by a Monte-Carlo approach [1]. The advantages of PF over different tracking methods such as Kalman filter are in its ability to use non-linear models and the ability to use non-Gaussian distributions. On the other hand, the disadvantage of the PF is its use of Monte Carlo as the performance of the PF strongly depends on the number of particles used. A large number of particles will simulate the required distributions more accurately leading to better results but also increase significantly the computational load. In many cases, weight computation of each particle can be computationally expensive. A common example for this case is object tracking in videos where the weight of each particle is determined by the Bhattacharyya coefficient [2] or by Earth-Moving-Distance (EMD) [3], which requires to evaluate histograms over a large number of bins, especially when color is involved. When the number of particles is either moderate or large (typically a few thousands) the computational load becomes a serious bottleneck to achieve real-time processing.

In this work, we propose a new method to evaluate particles weights using multiscale function extension (MSE) algorithm [4]. The MSE approach consists of two steps: subsampling and extension. In the subsampling step, the particles are sampled to achieve maximal coverage using a small fraction of the actual number of particles with respect to their density. This is done by a special type of matrix decomposition called Interpolative-Decomposition(ID). Then, the weights are computed for this small set of particles. In the next step (extension), the weights are extracted for the rest of the particles using the MSE method. The method uses coarse-to-fine hierarchy of the multiscale decomposition of a Gaussian kernel that represents the similarity between the particles. This generates a sequence of subsamples, which we refer to as adaptive grids, and a sequence of approximations to a given empirical function on the data, as well as their extensions to any missing multidimensional data point. Since in many cases the computational load of the weights is heavy, this approach can reduce the computational load significantly and accelerate the PF, allowing us to use more particles. Increasing the number of particles is needed since many of today tasks are geared to track objects "buried" in huge data streams such as video, communication and telemetric data.

Particle filters were studied in many works and used in several applications domains such as computer vision, robotics, target tracking and finance. While PF can be robust to both the input observations distribution and the behavior of the additive noise, its implementation is computationally intensive. Making it working in real-time (computationally efficient) has become a major challenge when objects tracking is done in high dimensional state space, or when dealing with multiple target tracking. Comprehensive tutorials and surveys on the different variations and recent progress in PF methods are given in [1], [5].

II. Particle Filter Algorithm

In general, PF is a model estimation technique based on simulation that uses Monte Carlo methods for solving a recursive Bayesian filtering problem [1]. It is used for estimating the state x_n at time n from a noisy observations $y_1, ..., y_n$. A dynamic state space equations are used for modeling and prediction. The basic idea behind PF is to use a sufficiently large number of "particles". Each particle is an independent random variable which represents a possible state of the target. For example, a state can be a location and velocity. In this case, each particle represents a possible location and velocity of the target from a proposed distribution. The system model is applied to the particles in order to perform prediction to the next state. Then, each particle is assigned a weight, which represents its reliability or the probability that it represents the real state of the target. The actual location (the output

of the PF) is usually determined as the maximal likelihood of the particle's distribution. The algorithm robustness and accuracy are determined by the number of computed particles. A large number of particles is more likely to cover a wider state subspace in the proximity of the target, as well as a better approximation of the state distribution function. However, the cost of such improved tracking produces higher computational load since each particle needs to be both advanced and weighted while this is repeated in each cycle.

III. Multiscale Function Extension

Given a set of N particles $\mathcal{P}_N = \{p_1, p_2, \ldots, p_N\}$ and their mutual distances, we wish to estimate the value of their weight function through a small subset $\mathcal{P}_n$ of n particles ($n < N$ is a predefined number), for which we compute the weights directly. Formally, our goal is to interpolate the weight function $W : \mathcal{P}_n \to \mathbb{R}$ to $\mathcal{P}_N$, given a distance function $d : \mathcal{P}_N \times \mathcal{P}_N \to R$. For that purpose we use the multiscale function extension (MSE) [4], which is a multiscale, numerically stable interpolation method.

Each scale of the MSE is divided into two phases - a subsampling phase and an extension phase. The first phase is done by a special decomposition, known as interpolative decomposition (ID), of an affinities matrix associated with $\mathcal{P}_n$. The second phase extends the function from $\mathcal{P}_n$ to $\mathcal{P}_N$, using the output of the first (sampling) phase. The essentials of the MSE are describe in sections III-A and III-B. For further reading we refer the reader to [4].

We use the following notation: s denotes the scale parameter, $s = 0, 1, \ldots$, $\epsilon_s = 2^{-s}\epsilon_0$ for some positive number ϵ_0, and $g^{(s)}(r) = \exp\{-r^2/\epsilon_s\}$. For a fixed scale s we define the function $g_j^{(s)} : \mathcal{P}_N \to \mathbb{R}$, $g_j^{(s)}(p) = g^{(s)}(d(p_j, p))$ to be the Gaussian of width ϵ_s, centered at p_j. $A^{(s)}$ is the $n \times n$ affinities matrix associated with $\mathcal{P}_n$, whose (i, j)-th entry is $g^{(s)}(d(p_i, p_j))$. Note that the j-th column of $A^{(s)}$ is the restriction of $g_j^{(s)}$ to $\mathcal{P}_n$. $\mathcal{P}_n^c$ is the complementary set of $\mathcal{P}_n$ in $\mathcal{P}_N$. The spectral norm of a matrix A is denoted by $\|A\|$, and its j-th singular value (in decreasing order) is denoted by $\sigma_j(A)$.

A. Data subsampling through ID of Gaussian matrix

Let s be a fixed scale. Our goal is to approximate W by a superposition of the columns of the affinities matrix $A^{(s)}$, then to extend W to $p_* \in \mathcal{P}_n^c$, based on the affinities between p_* and the elements of $\mathcal{P}_n$. At first sight, we could solve the equation $A^{(s)}c = W$ and, using the radiality of $g^{(s)}$, to extend W to p_* by $\hat{W}(p_*) = \sum_{i=1}^{n} c_i g_j^{(s)}(p)$, which is exact on $\mathcal{P}_n$, i.e. $\hat{W}(p_j) = W(p_j), j = 1, 2, \ldots, n$. This method is known as Nyström extension [6], [7]. As proved in [4], the condition number of $A^{(s)}$ is big for small values of s, namely $A^{(s)}$ is numerically singular. On the other hand, too big s would be resulted in a short distance interpolation. Moreover, even if we would choose such s for which $A^{(s)}$ is numerically non-singular and the interpolation is not for too short distance, interpolation by a superposition of translated Gaussian of fixed width, would not necessarily fit the nature of W.

In order to overcome the numerical singularity of $A^{(s)}$, we apply an interpolative decomposition (ID). The deterministic version of the ID algorithm can be found in [4], whose complexity is $\mathcal{O}(mn^2)$, and a randomized version can be found in [8]. The latter is based on random-projections and its complexity is $\mathcal{O}(k^2 n \log n)$. Since each column of $A^{(s)}$ corresponds to a single data point in $\mathcal{P}_n$, selection of columns subset from $A^{(s)}$ is equivalent for subsampling of $\mathcal{P}_n$ data points.

B. Multiscale function extension

Let $A^{(s)} = B^{(s)}P^{(s)}$ be the ID of $A^{(s)}$, where $B^{(s)}$ is an $n \times k$ matrix, whose columns constitute a subset of the columns of $A^{(s)}$, and let $\mathcal{P}^{(s)} = \{p_{s_1}, \ldots, p_{s_k}\}$ the associated sampled dataset. The extension of W to $\mathcal{P}_n^c$ is done by orthogonally projecting W on the columns space of $B^{(s)}$, and extending the projected function to $\mathcal{P}_n^c$ in a similar manner to Nyström extension method, using the radiality of $g^{(s)}$. The single scale extension (SSE) algorithm can be found in [4] (Algorithm 3), whose complexity is $\mathcal{O}(nk^2)$.

Obviously, $w^{(s)}$ is not necessarily equal to w, namely the output of the SSE algorithm is not an interpolant of w. In this case we apply the SSE algorithm once again to the residual $w - w^{(s)}$ with narrower Gaussian, that ensures a bigger numerical rank of the next-scale affinities matrix $A^{(s+1)}$ and, as a consequence, a wider subspace to project the residual on. Such approach is described in Algorithm 4 in [4]. We shell call this algorithm the multiscale extension (MSE) Algorithm, whose complexity is $\mathcal{O}(n^3)$.

IV. Multiscale Particle Filter (MSPF)

In order to accelerate the PF computation, while executing it with a large number of particles, we will apply an intelligent sampling of the particles, followed by an extension method to compute the weights of the rest. This will allow us to compute a relatively small number of particle weights in each cycle. Such approach can be effective if the particle weight calculation is computationally expensive.

A. Particle Subsampling

In each cycle of the PF algorithm, we first resample a new set of N particles from the set $\mathcal{P} = \{x_t^{(n)}, w_t^{(n)}\}, n = 1, .., N$ using their weights as the distribution function. Once we apply the dynamic model on each particle and advance it, we need to compute their new weights. To do that, we first select a small subset of the particles. We wish to find a good set of particle candidates that will capture the geometry of the weight function $W : \mathcal{P}_n \to \mathbb{R}$. To find such candidates, we define a distance metric between the particles. In our experiments we used the euclidean distance between each two particles viewed as vectors, but other metrics can be used as well. We select the particle candidates using the ID Algorithm described in Section III-A. We construct an affinity matrix $A^{(s)}$ containing the affinities between the particles, using a Gaussian kernel

that is based on the given distance metric $d(p_i,_p j)$ between the particles.

$$[A^{(s)}]_{ij} = \exp\left(\frac{-d(p_i,p_j)^2}{\epsilon_s}\right), i,j = 1,..,N. \qquad \text{(IV.1)}$$

We calculate the affinities for all the particles in $\mathcal{P}$ so $A^{(s)}$ is an $N \times N$ matrix defined by Equation IV.1. The number of candidates we wish to receive is at most k. The value k is usually selected according to the computation budget we have to calculate the weight function in each cycle. The output of the ID algorithm will be a set $\mathcal{P}_k$ of k particles selected from $\mathcal{P}$. We compute the weights of the k particles we selected, as we do in the original PF algorithm.

B. Weight Calculation using Function Extension

Now that we have a set of particles $\mathcal{P}_k = \{x_t^{(n)}, w_t^{(n)}\}, n = 1,..,k$ with their calculated weight values, we can continue and compute the weights of the rest of the particles. Using the MSE algorithm, we compute the weight value of each of the other $N-k$ particles, by using the set $\mathcal{P}_k$, and the first k columns of the affinity matrix $A_k^{(s)}$. These columns contains the affinities between each pair of particles in $\mathcal{P}_k$ and the affinities between particles in $\mathcal{P}_k$ and all other particles. The output of the MSE algorithm is the weights of the $N-k$ particles that were not selected in the previous step. The extension method allow us to avoid a direct computation of the weight for the rest of the $N-k$ particle. This is especially effective when we can not compute the weight for all particles if the observation has some missing data, or if the computation is too intense. Once we calculated all the weights we select the particle with the maximum likelihood (weight) as the prediction result and continue to the next cycle.

Algorithm IV.1: Multiscale Particle filter
Input: Initial state x_0 and current observations $y_1, ..., y_T$
Output: Estimated observations $x_1, ..., x_T$

1: Initialize weights $w_0^{(n)} = \frac{1}{N}$, and $x_0^{(n)} \sim p(x_0)$, $n = 1, ..., N$.
2: **for** time steps t=1,...,T **do**
3: Resample N new particles by their distribution determined by weights: $w_t^{(n)}$
4: Prediction: Apply the dynamic model on each particle to estimate next state using x_{t-1} and $y_1, ..., y_t$

$$\tilde{x}_t^{(n)} \sim q(x_t^{(n)} | x_{t-1}^{(n)}, y_1, ..., y_t)$$

5: Selection : Select a subset of size k out of the new particles $\tilde{x}_t^{(n)}$, by computing the affinity matrix $A^{(s)}$ and using the ID Algorithm.
6: Calculate weights of the k selected particles using

$$w_t^{(n)} \propto \frac{p(y_t|x_t^{(n)})p(x_t^{(n)}|x_{t-1}^{(n)})}{q(x_t|x_{t-1}^{(n)}, y_1, ..., y_t)}, n = 1, ..., N$$

7: Weight extension: Calculate the weights of the $N-k$ particles using the Multiscale Function Extension Algorithm (See Algorithm 4 in [4]).
8: Normalize weights:

$$\tilde{w}_t^{(n)} = \frac{w_t^{(n)}}{\sum_{i=1}^{N} w_t^{(i)}}$$

9: Set x_t to be the particle $\tilde{x}_t^{(i)}$ with maximum weight $\tilde{w}_t^{(i)} \geq \tilde{w}_t^{(n)}, n = 1, .., N$
10: **end for**

V. Experimental Results

To test the performance of Algorithm IV.1, we preformed several experiments of tracking objects in both synthetic and real videos, and comparing the results to other tracking methods.

A. Multiple Target Tracking

he MSPF Algorithm IV.1 was tested on a video sequence that contains multiple objects. In this case, the tracking was achieved by the application of two different PF types algorithms, where each had its own set of particles and a separate set of observations. Each particle describes a state of a single target. Another approach to track multiple objects is to create a "super-state" particle, which describes the state of all the objects inside the video sequence. In this case, the number of fields inside the particle vector was $n \times k$ where n is the number of targets and k is the number of parameters required to describe a single target. In the latter scenario, the MSE algorithm outperformed standard interpolation methods since it handled better data points in high dimensions. The advantage of using the "super-state" particle is by enabling to advance a particle state by dynamic model equations that took into account the state of all the objects within a particle including dependencies between objects.

In order to test the tracking performance using the "super-state" particle, we tracked two tennis players in a video sequence. The players are represented by a single particle with $6 \times 2 = 12$ coordinates, 6 for each player (location in x and y, velocity in x and y, width and hight). In each algorithm cycle, the prediction step advanced the particles by the application of the model equations separately to each coordinate. The weight calculation was done in each region separately and then multiplied the Bhattacharyya coefficient to obtain a single weight. Then, the extension step was applied as before using the weighted Euclidean metric for each particle that has 12 coordinates. By using Algorithm IV.1, we were able to track both targets successfully with the lowest computational cost in comparison to other extension methods that are based on standard interpolation such as B-splines, cubics and nearest neighbor. We used 1500 particles to track both players. In each step of the algorithm, we calculated the weights for 150 selected particles and interpolated the weights for the other 1350 particles by using the MSE method. The complete videos of the basketball and tennis games tracking can be seen in our

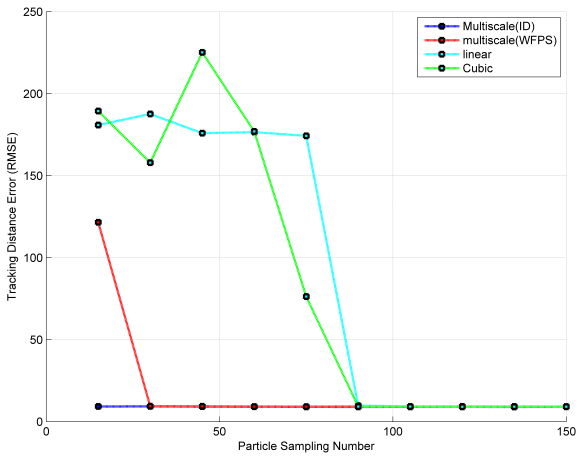

Fig. V.1. Comparing the RMSE between different methods - multiscale with ID sampling , Multiscale with WFPS sampling, linear approximation, cubic approximation.

website[1].

B. Comparison with Other Approximation Methods

In order to compare the proposed method with different approximation methods, we applied the PF algorithm and in each run we used a different approximation method to calculate the particle's weights. In this comparison, we used a synthetic movie. We generated a video sequence by moving a colored disc over a still image. The disc moved along a non-linear parametric function. This allows us to know the ground truth of the target at any frame. We applied the PF algorithm to the synthetic video sequence several times, each with different interpolation method. We compared the total Root Mean Square Error (RMSE) for each approximation method measured on the distance between the estimation and the real location of the target. The MSPF Algorithm IV.1 had the lowest error rate even when we use a sampling factor between 2%-5% from the total number of particles. When such subsampling factor was used, all the other tested methods fail (error grew).

Overall, the MSPF Algorithm IV.1 achieved the lowest computational time while maintaining a low error rate.

C. Comparison with the EMD Measurement

Recently, the Earth Moving Distance (EMD) [3] was used for particles weight computation since this particle weight fits deformable objects [9]. We tested Algorithm IV.1 with the EMD metric to demonstrate how well the extension scheme fits it. Several runs were conducted on the "Lemming" sequence from the PROST database. Weights were calculated for 10% from the total number of particles while the rest of the particles were estimated using the MSE Algorithm.

Table V.1 shows the time differences between the naive version of the PF algorithm that uses the EMD metric and our implementation that uses the MSE Algorithm. For the latter, 10% of the particles were sampled, and the MSE was applied to the other 90% of the particles.

[1] http://www.cs.tau.ac.il/~yanivshm/mspf

TABLE V.1
EMD ACCELERATION TIME COMPARISONS IN [SEC]. SAMPLING WAS 10% FROM THE TOTAL NUMBER OF PARTICLES.

# of Particles	Time [no MSE]	Time [with MSE]	Acceleration Factor
2000	63	10.6	5.9
4000	125	32	3.9
6000	187	75.4	2.5
8000	260	151	1.7
10000	294	266	1.1

VI. CONCLUSION

In this work, we presented several contributions. We reduced the PF computational time by applying a novel multiscale extension (MSE) method to reduce the particle weight calculation. This allowed us to use more particles within a given computational budget thus improving the performance of the PF. We tested our modified PF algorithm on real video sequences to track a single and multiple targets, and compared it with other extension methods.

ACKNOWLEDGMENT

This research was partially supported by the Israel Science Foundation (Grant No. 1041/10) and by the Israeli Ministry of Science & Technology (Grant No. 3-909).

REFERENCES

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

[2] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc*, vol. 35, no. 99-109, p. 4, 1943.

[3] Y. Rubner, C. Tomasi, and L. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[4] A. Bermanis, A. Averbuch, and R. Coifman, "Multiscale data sampling and function extension," *Applied and Computational Harmonic Analysis*, vol. http://dx.doi.org/10.1016/j.acha.2012.03.002, 2012.

[5] A. Doucet and A. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, pp. 656–704, 2009.

[6] C. Baker, *The numerical treatment of integral equations*. Clarendon press Oxford, 1977, vol. 13.

[7] B. Flannery, W. Press, S. Teukolsky, and W. Vetterling, "Numerical recipes in c," *Press Syndicate of the University of Cambridge, New York*, 1992.

[8] P. Martinsson, V. Rokhlin, and M. Tygert, "A randomized algorithm for the decomposition of matrices," *Appl. Comput. Harmon. Anal.*, 2010.

[9] S. Avidan, D. Levi, A. Bar-Hillel, and S. Oron, "Locally orderless tracking," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 1940–1947.