This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

Accelerating Particle Filter using Randomized Multiscale and Fast Multipole Type Methods

Gil Shabat, Yaniv Shmueli, Amit Bermanis and Amir Averbuch

Abstract—Particle filter is a powerful tool for state tracking using non-linear observations. We present a multiscale based method that accelerates the tracking computation by particle filters. Unlike the conventional way, which 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 a particle subset to recover the density function for all the rest of the particles not included in the chosen subset. The computational effort is substantial especially when multiple objects are tracked concurrently. The proposed algorithm significantly reduces the computational load. By using the Fast Gaussian Transform, the complexity of the particle selection step is reduced to a linear time in n and k , where n is the number of particles and k is the number of particles in the selected subset. We demonstrate our method on both simulated and on real data such as object tracking in video sequences.

Index Terms—particle filter, multiscale methods, nonlinear tracking, fast multipole method

1 Introduction

Particle filter (PF) is a powerful tool for state tracking, based on non-linear observations, that uses the Monte-Carlo approach [1].

PF implements a recursive Bayesian filter where the probability density function (PDF) is represented by a set of random samples (particles) and not in its analytical form. The number of particles controls the approximation accuracy. A large number of particles will lead to a more accurate representation of the functional form of the PDF. The particles are propagated and advanced under the control of the system dynamics and under target measurement model. In the Sequential Importance Resampling (SIR) version of the PF, particles are resampled at each cycle by using importance sampling on their probabilities that are called "weights". The particle with the maximal likelihood is selected to be the current predicted state of the target. Unlike Kalman filters, PFs are not restricted by stationary linear-Gaussian assumptions that make them more robust and suitable for a larger set of problems. Although the PF concept is fairly straightforward, it becomes computationally expensive for practical implementations, as a very large number of particles are needed in order to accurately approximate the PDF of the observed target(s). The increase in computational power, in the last decades, have enabled to introduce PF based solutions to real world problems.

The role of PF acceleration is to increase the number of propagated particles while maintaining the same computational cost. A large number of particles represent the required distributions more accurately that leads to better results. Many systems are geared to track objects that are "buried" in huge data-streams such as video sequences, communication and telemetric data. Predicting the next object state or tracking it has to be done in near real-time especially by devices that have limited computational resources such as embedded devices.

In this work, we develop an improved PF algorithm that avoids the need to compute the likelihood function (the weights) for all the particles. The particle weight computation is expensive in cases such as tracking objects in video sequences. Instead, the weights are computed for only a subset of the particles and their values are used to estimate the weights for the rest of them. To select a representative set of particles, we use the interpolative decomposition (ID) method [2]. Then, the weights are extended to the rest of the particles using a multiscale function extension (MSE) method [3], which is an application of the Nyström extension method ([4], [5]) to particles not in the selected set. The extension is based on the similarities between particles that are not in the selected subset and the particles in the selected subset. The MSE uses radial Gaussian functions with varying scales to estimate the coefficients of the Nyström extension. The MSE method is shown in [3] to be both accurate and numerically stable overcoming the deficiencies in the Nyström method.

Interpolative decomposition (ID) is a method to approximate a matrix by selecting a set of k independent columns that constitute a basis. We use the ID method to select particles that best represent the PDF. To find this particle subset we compute an affinity matrix A for the particles and apply the ID algorithm to compute a k independent set of columns from A. These columns correspond to the most relevant particles.

Once the particle selection process is completed, the weights are computed only for the selected particles. The weight values are extended to the rest of the particles by the application of MSE. The motivation for using MSE, as our extension method, is based on the fact that the PDF is generally a smooth function and that the

G. Shabat, Y. Shmueli, A. Bermanis and A. Averbuch are from Tel Aviv University

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

MSE method is strongly related to a Gaussian process regression (GPR) [6], which is an extension method in the field of statistical inference. We use a randomized implementation of the ID algorithm, which is based on random projections, to minimize the computational cost of selecting the most relevant particles [7]. To reduce the computational cost even further, we propose a selection algorithm that is based on the Farthest Point Sampling (FPS) [8] method combined with density estimation. We refer to our method, which combines the FPS algorithm with density estimator, as a weighted FPS (WFPS). The density estimator was implemented using the Fast Multipole Method (FMM) [9] that asymptotically has a lower computational cost.

In our experiments, we were able to accelerate the weight calculation step to be approximately 10 times faster compared to the standard particle filter. The running time and tracking error of the algorithm were compared to the standard PF (sequential importance resampling) as well as to other weight interpolation methods indicating that the proposed algorithm was able to maintain the same error rate with a much shorter running time.

This paper is organized as follows: Section 2 presents related work on PF acceleration. Section 3 describes the PF algorithm, explains its limitations and how it can be used for object tracking in video sequences. Section 4 describes the multiscale sampling and extension techniques with its mathematical tools such as the randomized interpolative decomposition (ID). Section 5 presents the full multiscale PF algorithm that accelerates the standard PF. An additional acceleration is achieved by the algorithm in Section 6 to overcome the multiscale PF scalability bottleneck. Experimental results are presented in Section 7. We compare between the performances of the presented method and other methods in different scenarios.

2 Related Work

PFs have been studied in many works and used in different application domains such as computer vision, robotics, target tracking and finance. Example applications include hand gesture-based interface [10], real-time tracking of soccer players [11], mobile robot localization [12] and visual tracking of human face [13]. The PF employs a sequential Monte Carlo approach to solve recursive Bayesian filtering problems. The Monte Carlo sampling method combined with the Bayesian inference enables the PF to provide a solution for non-linear and non-Gaussian problems. While PF can be robust to both the input observations distribution and to the observations noise level, its implementation is computationally expensive. Making it to work 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 targets tracking. Such instances require to use more particles and thus the problem quickly becomes intractable. In addition, due to the nature of the PF algorithm and its repeated sampling method, the algorithm may suffer from sample degeneracy where most of the particles have negligible weights. Over the last two decades, different variations of the PF algorithm have emerged to overcome these limitations. Methods such as Auxiliary PF [14], Gaussian Sum PF [15], Unscented PF [16] and Swarm Intelligence based PF [13] were developed in order to overcome these limitations by improving the underlining sampling techniques and by providing better evaluations to the proposal distribution of the particles in each step of the algorithm.

The problem of tracking curves in a dense visual clutter is investigated in [17], where a method for learning the dynamical models using visual observations and propagate a randomly generated set over time to achieve near real time tracking is introduced. High dimensional models for tracking people using Monte Carlo filtering and hybrid hypothesis methods are also studied in [18], [19], [20], [21]. By using randomization methods for improving the particles re-sampling and applying specific assumptions to the dynamical models, efficient visual tracking is achieved. Overcoming the uncertainty induced by occlusion, abrupt motion or appearance changes while still preventing sample impoverishment problems is demonstrated in [22], [23].

Additional methods for estimating the posterior densities were suggested in [24], [25], [26], [27]. These methods propose effective ways for representing the posterior density that result in using fewer particles. The unscented PF (UPF), for example, defines points that capture sufficient distribution statistics. Then, it propagates each particle and incorporates the new observation to produce a Gaussian estimation of its proposal distribution. By using the input observations, the UPF can achieve more accurate proposal distribution estimation than what the regular PF implementation achieves. In continuous density propagation [27], density approximation and interpolation techniques are employed to represent the proposal distribution efficiently. The density functions are represented by Gaussian mixtures where the number of components, their coefficients and other related statistical parameters are automatically determined by the algorithm. Similar sampling and approximation methods for accelerating nonparametric belief propagation (NBP) were also developed in [26]. The core of the NBP algorithm requires a repeated sampling from products of Gaussian mixture, which makes the algorithm computationally expensive. To accelerate the process, Gaussian mixture density approximation using mode propagation and kernel fitting is applied. The products of the Gaussian mixture are approximated accurately by just a few mode propagations and kernel fitting steps. This significantly accelerates the sampling method since it uses a fewer samples. The approach presented in this paper is similar in that sense, but unlike the above methods we do not use fewer particles to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

accelerate the PF. Instead, we compute the weights for only a particle subset. This subset is selected using matrix decomposition methods and the remaining weights are iteratively interpolated using multiscale extension methods which are accurate and numerically stable. The advantage of computing the weights for only a subset of the particles, enables generating a large set of particles and more freedom in choosing an appropriate subset for the estimation of the distribution, which leads to a more accurate estimation and therefore better performance.

The challenge, which all these methods face, is how to incorporate a new observation into the estimated proposal distribution function of the particles while maintaining a reasonable computation cost. Comprehensive tutorials and surveys on different PF versions and recent advances in PF methods are given for example in [1], [28], [29].

3 Particle Filter Algorithm

PF is an on-line density estimation technique based on target simulation that uses Monte Carlo methods for solving a recursive Bayesian filtering problem. PF is used to estimate the state x_i at time i from noisy observations $y_1, ..., y_i$. Dynamic state space equations are used for modeling and prediction of the target state. The basic idea behind the PF approach is to use a sufficiently large number of "particles". Each particle is an independent random variable which represents a possible target state. For example, a state can be location and velocity. In this case, each particle represents a possible location and a possible velocity of the target from a proposal distribution. The system model is applied to the particles in order to predict the next state. Then, each particle is assigned a weight, which represents its reliability or the probability that it represents the real target state. The actual location (the output of the PF algorithm) is usually determined as the maximum a posteriori probability of the particle's posterior distribution. The algorithm robustness and accuracy are determined by the number of participating particles. A large number of particles is more likely to cover a wide state subspace in the proximity of the target, as well as a better approximation of the state distribution function. However, the computational cost of such an improved tracking is high since each particle needs to be both advanced in time and weighted. This is repeated in each cycle of the algorithm.

A description of the PF algorithm flow is given by Algorithm 3.1. Assume that p represents the *proposal distribution* that is used to predict the next particles states. The optimal proposal distribution is the target's distribution, which is given by $p(x_k|x_{k-1}, y_k)$. Since this computation is impractical, an estimated distribution, which is called the proposal distribution, is used. In our case, this distribution is computed by utilizing the evolution of the system (a physical model) to the particles.

The weights computation in Algorithm 3.1, step 5, can be expensive in some cases. For example, when the

Algorithm 3.1: Particle Filter (SIR)
Input: n number of particles; x_0 initial state; $y_1, ..., y_T$ current observations; $q(\cdot)$ proposal distribution function; $p(\cdot)$ approximated posterior distribution function
Output: $x_1, ..., x_T$ estimated observations

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

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

5: Weights calculation:

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

6: Weights normalization:

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

7: x_t is set to be the particle $\tilde{x}_t^{(l)}$ where $l = \underset{1 \le i \le n}{\arg\max}\ w_t^{(i)}$.
8: **end for**

PF algorithm is used for tracking a target in videos, it is common to use distances between histograms for weighting the measurements. An RGB image with 256 gray-levels for each pixel will have a histogram of $256^3 = 16,777,216$ bins. Then, the distance between two histograms $\mathbf{h_1}$ and $\mathbf{h_2}$ (both vectors have length of size the number of bins) can be measured, for example, by the Bhattacharyya coefficient

$$B = \sqrt{\mathbf{h_1}^T \mathbf{h_2}}. \tag{3.1}$$

The color histogram calculation complexity for a given particle depends on the number of bins we use and the number of pixels we need to iterate, as each particle points to an image patch of the target. Assume that the number of bins is b and the number of particles is n. The total weight calculation complexity in each cycle can be very expensive for large b and n values. On the other hand, a large number of bins can help in improving the distance accuracy that affects the weights estimations.

Despite the robustness of the PF algorithm, it suffers from several limitations. Usually, each new observation requires some preprocessing followed by a weight calculation of each particle. Both steps can be computationally expensive in applications such as computer vision and robotics. In such problems, the observation contains a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

large amount of data. For example, when tracking a target within a video sequence, each measurement consists of an image frame that may contain several millions of pixels (as in HD format). In addition, each particle is assigned a weight that is based on some calculation applied to a subset of the measured data. This subset can be relatively large (for example, an image patch with thousands of pixels). As the number of particles becomes large, the total computational load can become extremely expensive. When dealing with high dimensional particles that contain many parameters in each state, the number of needed particles increases exponentially to cover a region of interest around the current target state. However, using a large number of particles is important to obtain high diversity for the particles so they will represent the solution space adequately.

4 Multiscale Function Extension Method

Given a set $\mathcal{P}_n = \{p_1, p_2, \ldots, p_n\}$ of n particles, we want to estimate the values of their weights using a small subset $\mathcal{P}_k \subseteq \mathcal{P}_n$ of k particles. Here, $k \ll n$ is a predefined number for which the weights of $\mathcal{P}_k$ are computed directly. Formally, our goal is to interpolate the weight function $w : \mathcal{P}_k \to \mathbb{R}$ from $\mathcal{P}_k$ to $\mathcal{P}_n$ (as calculated in Step 5 in Algorithm 3.1). For that purpose, we use the MSE method [3], which is a multiscale based algorithm. The MSE is an iterative method. Each MSE iteration contains two phases: subsampling and extension. The first phase is done by a special decomposition, known as interpolative decomposition (ID) [2], of an affinities matrix associated with $\mathcal{P}_k$. The second phase extends the function from $\mathcal{P}_k$ to $\mathcal{P}_n$ using the output from the first (sampling) phase. The essentials of the MSE are described in Sections 4.1, 4.2 and in [3].

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

$$g^{(s)}(r) \triangleq \exp\{-r^2/\epsilon_s\}. \tag{4.1}$$

For a fixed scale s, we define the functions $g_j^{(s)} : \mathcal{P}_n \to \mathbb{R}$, $j = 1, \ldots, n$

$$g_j^{(s)}(p) \triangleq g^{(s)}(d(p_j, p)) \tag{4.2}$$

to be a Gaussian of width ϵ_s centered at p_j. Here, $d : \mathcal{P}_n \times \mathcal{P}_n \to \mathbb{R}$ is a distance function defined among the particles. For example, it can be the Euclidean distance between their coordinates. Without loss of generality, we assume that the chosen particles are indexed such that $\mathcal{P}_k = \{p_1, \ldots, p_k\}$. Let $A^{(s)}$ be the $k \times k$ affinities matrix associated with $\mathcal{P}_k$, whose (i, j)-th entry is $g^{(s)}(d(p_i, p_j))$. In other words,

$$A^{(s)}(i, j) \triangleq g^{(s)}(d(p_i, p_j)), \quad i, j = 1, \ldots, k. \tag{4.3}$$

Note that the j-th column of $A^{(s)}$ is the restriction of $g_j^{(s)}$ to $\mathcal{P}_k$. $\mathcal{P}_k^c$ is the complementary set of $\mathcal{P}_k$ 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 $\sigma_j(A)$. $\mathbf{w} = (w_1, w_2, \ldots, w_k)^T$ are the values of the weight function w on the particles in $\mathcal{P}_k$, where w_j is the weight of p_j.

4.1 Data Subsampling Through ID of a Gaussian Matrix

Algorithm 4.1: Deterministic interpolative decomposition

Input: An $m \times n$ matrix A and an integer k, s.t. $k < \min\{m, n\}$.

Output: An $m \times k$ matrix B, whose columns are a subset of A's columns, and a $k \times n$ matrix P s.t. $\|A - BP\| \leq \sqrt{4k(n-k)+1}\,\sigma_{k+1}(A)$

1: Apply a pivoted QR algorithm to A (Algorithm 5.4.1 in [30]),

$$AP_R = QR,$$

where P_R is an $n \times n$ permutation matrix, Q is an $m \times m$ orthogonal matrix and R is an $m \times n$ upper triangular matrix, where the diagonal absolute values are decreasingly ordered.

2: Split R and Q s.t.

$$R \triangleq \left[\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & R_{22} \end{array}\right], \quad Q \triangleq \left[\begin{array}{c|c} Q_1 & Q_2 \end{array}\right]$$

where R_{11} is $k \times k$, R_{12} is $k \times (n-k)$, R_{22} is $(m-k) \times (n-k)$, Q_1 is $m \times k$ and Q_2 is $m \times (m-k)$.

3: Define the $m \times k$ matrix

$$B \triangleq Q_1 R_{11} \tag{4.4}$$

4: Define the $k \times n$ matrix

$$P \triangleq \left[\begin{array}{c|c} I_k & R_{11}^{-1} R_{12} \end{array}\right] P_R^T$$

where I_k is the $k \times k$ identity matrix.

Let s be a fixed scale. Our goal is to approximate w by a superposition of the columns in the affinity matrix $A^{(s)}$, then to extend w to $p_* \in \mathcal{P}_k^c$ based on the affinities between p_* and the elements of $\mathcal{P}_k$. Due to Bochner's theorem, as long as $\mathcal{P}_k$ consists of k distinct particles, $A^{(s)}$, which is defined in Eq. 4.3, is strictly positive definite. At first sight, we can solve the equation $A^{(s)}c = \mathbf{w}$ and by using the radiality of $g^{(s)}$ (defined in Eq. 4.1), to extend w to p_* by $w(p_*) = \sum_{i=1}^k c_i g_i^{(s)}(p_*)$ (defined in Eq. 4.2), which is exact on $\mathcal{P}_k$. That is, $w_j = w(p_j), j = 1, 2, \ldots, k$. This method is known as the Nyström extension [4], [5]. As proved in [3], the condition number of $A^{(s)}$ is large for small values of s, namely $A^{(s)}$ is numerically singular. On the other hand, a too big s results in a short distance interpolation. Moreover, even if we choose such s for which $A^{(s)}$ is numerically nonsingular and the interpolation is not for a too short distance, interpolation by a superposition of translated Gaussian of a fixed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

width will not necessarily fit the properties of w. In order to overcome the numerical singularity of $A^{(s)}$, we apply the ID procedure to $A^{(s)}$.

An ID of order k of an $m \times n$ matrix A consists of an $m \times k$ matrix B whose columns consist of a subset of the columns of A, as well as a $k \times n$ matrix P, such that a subset of the columns of P constitutes a $k \times k$ identity matrix, and $A \approx BP$ in the sense that $\|A - BP\| \lesssim \mathcal{O}(n, \sigma_{k+1}(A))$. Usually, k is chosen to be the numerical rank of A up to a certain accuracy $\delta > 0$, i.e. $k = \#\{j : \sigma_j(A) \geq \delta\sigma_1(A)\}$. This selection of k guarantees that the columns of B constitute a well conditioned basis to the range of A, whose condition number is of order δ. The deterministic ID algorithm is described in Algorithm 4.1 whose complexity is $\mathcal{O}(mn^2)$.

Additionally to Algorithm 4.1, there are randomized versions of the ID algorithm that require less computational operations. For example, Algorithm 4.2 is a random-projections based algorithm [7]. It produces an ID for a general matrix $m \times n$ matrix A and an integer $l < \min\{m, n\}$, s.t.

$$\|A - BP\|_2 \lesssim l\sqrt{mn}\sigma_{l+1}(A). \tag{4.5}$$

The complexity is $lC_A + lC_{A^T} + \mathcal{O}(l^2 n \log(n))$, where C_A is the cost of applying A to a vector of length k, and C_{A^T} is the cost of applying A^T to a vector of length m. Algorithm 4.2 uses the deterministic ID Algorithm 4.1 by applying it to a smaller matrix than A.

Each column of $A^{(s)}$, as defined in Eq. 4.3, corresponds to a single particle in $\mathcal{P}_k$. The columns subset selection from $A^{(s)}$ is equivalent to $\mathcal{P}_k$ particles that are subsampled from the associated $\mathcal{P}_n$ particles.

4.2 Multiscale Function Extension Algorithm

Let $A^{(s)} \approx B^{(s)}P^{(s)}$ be the ID of $A^{(s)}$, where $B^{(s)}$ is a $k \times r$ matrix, whose columns constitute a subset of the columns of $A^{(s)}$, and $\mathcal{P}^{(s)} = \{p_{s_1}, \ldots, p_{s_r}\}$ is its associated sampled dataset. The extension of the weight function w from $\mathcal{P}_k$ to $\mathcal{P}_k^c$ is done by an orthogonal projection of $\mathbf{w}$ on the columns space of $B^{(s)}$, and by extending the projected function to $\mathcal{P}_k^c$ in a similar manner to Nyström extension method that uses the radiality of $g^{(s)}$. Algorithm 4.3, whose complexity is $\mathcal{O}(kr^2)$, describes the single-scale extension algorithm.

Since the columns of $B^{(s)}$ do not necessarily constitute a basis of $\mathbb{R}^k$, $\mathbf{w}^{(s)}$ is not necessarily equal to $\mathbf{w}$, namely the output of Algorithm 4.3 is not an interpolant of $\mathbf{w}$. This phenomenon is illustrated in Fig. 5.1 in [3]. In this case, we apply Algorithm 4.3 once again to the residual $\mathbf{w} - \mathbf{w}^{(s)}$ with a narrower Gaussian. This guarantees that the next-scale affinities matrix $A^{(s+1)}$ has a bigger numerical rank then $A^{(s)}$. As a consequence, it guarantees a wider subspace to project the residual on. The above is summarized in Algorithm 4.4 whose complexity is $\mathcal{O}(k^3)$.

Algorithm 4.2: Randomized interpolative decomposition
Input: An $m \times n$ matrix A and two integers $l < k$, s.t. $k < \min\{m, n\}$ (for example, $k = l + 8$).
Output: An $m \times l$ matrix B and an $l \times n$ matrix P that satisfy Eq. 4.5.

1: Use a random number generator to form a real $k \times m$ matrix G whose entries are i.i.d Gaussian random variables of zero mean and unit variance. Compute the $k \times n$ product matrix
$$W = GA.$$
2: Using Algorithm 4.1, form a $k \times l$ matrix S, whose columns constitute a subset of the columns of W and a real $l \times n$ matrix P, such that
$$\|SP - W\|_2 \leq \sqrt{4l(n-l)+1}\sigma_{l+1}(W).$$
3: From Step 2, the columns of S constitute a subset of the columns of W. In other words, there exists a finite sequence $i_1, i_2, \ldots, i_l$ of integers such that, for any $j = 1, 2, \ldots, l$, the j-th column of S is the i_j-th column of W. The corresponding columns of A are collected into a real $m \times l$ matrix B, such that, for any $j = 1, 2, \ldots, l$, the j-th column of B is the i_j-th column of A. Then, the sampled dataset is $D_s = \{x_{i_1}, x_{i_2}, \ldots, x_{i_l}\}$.

Algorithm 4.3: Single-scale extension
Input: Scale parameter s, $k \times r$ matrix $B^{(s)}$, the associated sampled dataset $\mathcal{P}^{(s)} = \{p_{s_1}, \ldots, p_{s_k}\}$, a new data point $p_* \in \mathcal{P}_k^c$, and the weight function $w : \mathcal{P}_k \to \mathbb{R}$ to be extended.
Output: The projection $\mathbf{w}^{(s)} = (w_1^{(s)}, w_2^{(s)}, \ldots w_k^{(s)})^T$ of $\mathbf{w} = (w_1, w_2, \ldots, w_k)^T$ on $B^{(s)}$ and its extension $\mathbf{w}_*^{(s)}$ to p_*.

1: Solve the least squares problem $\min_{c \in \mathbb{R}^r} \|B^{(s)}c - \mathbf{w}\|_2$ for $c = (c_1, c_2, \ldots, c_r)^T$.
2: Calculate the orthogonal projection of $\mathbf{w}$ on the columns of $B^{(s)}$, $\mathbf{w}^{(s)} = B^{(s)}c$.
3: Calculate the extension $\mathbf{w}_*^{(s)}$ of $\mathbf{w}^{(s)}$ to p_* using Eq. 4.2:
$$\mathbf{w}_*^{(s)} \triangleq \sum_{j=1}^{r} c_j g_{s_j}^{(s)}(p_*). \tag{4.6}$$

5 Multiscale Particle Filter (MSPF)

In order to accelerate the PF algorithm when it runs on a large number of particles, we apply particles subsampling. We use the MSE method to compute the weights for the rest of the particles. This will allow us to compute a relatively small number of particle weights in each cycle of the algorithm. This approach can be effective if the particle's weight calculation on all particles is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

Algorithm 4.4: Multiscale data sampling and function extension
Input: A dataset of k particles $\mathcal{P}_k = \{p_1, \ldots, p_k\}$, a positive number ϵ_0, a new particle $p_* \in \mathcal{P}_k^c$, a weight function $w : \mathcal{P}_k \to \mathbb{R}$, to be extended (represented by $\mathbf{w} = (w_1, w_2, \ldots, w_k)^T$ where w_j is the weight of p_j), and an error parameter $err \geq 0$.
Output: An approximation $\hat{\mathbf{w}} = (\hat{w}_1\ \hat{w}_2\ \ldots\ \hat{w}_k)^T$ of $\mathbf{w}$ on $\mathcal{P}_k$ that satisfies $\|\mathbf{w} - \hat{\mathbf{w}}\| \leq err$ and its extension $\hat{\mathbf{w}}_*$ to p_*.

1: Set the scale parameter $s = 0$, the approximation to $\mathbf{w}$, $\hat{\mathbf{w}} = 0$, and the extension of $\hat{\mathbf{w}}$ to p_*, $\hat{\mathbf{w}}_* = 0$.
2: **while** $\|\mathbf{w} - \hat{\mathbf{w}}\| > err$ **do**
3: Form the Gaussian affinities matrix $A^{(s)}$ (Eq. 4.3) on $\mathcal{P}_k$, with $\epsilon_s = 2^{-s}\epsilon_0$.
4: Set r to be the numerical rank of $A^{(s)}$ (see Definition 3.1 in [3]).
5: Apply Algorithm 4.1 to $A^{(s)}$ with the parameter r to get an $k \times r$ matrix $B^{(s)}$ and the associated sampled dataset $\mathcal{P}^{(s)}$.
6: Apply Algorithm 4.3 to $B^{(s)}$, $\mathcal{P}^{(s)}$, p_*, and $\mathbf{w} - \hat{\mathbf{w}}$. We get the approximation $\hat{\mathbf{w}}^{(s)}$ to $\mathbf{w} - \hat{\mathbf{w}}$ at scale s, and its extension $\hat{\mathbf{w}}_*^{(s)}$ to p_*.
7: Accumulate the approximations from step 6: Set $\hat{\mathbf{w}} = \hat{\mathbf{w}} + \hat{\mathbf{w}}^{(s)}$ and $\hat{\mathbf{w}}_* = \hat{\mathbf{w}}_* + \hat{\mathbf{w}}_*^{(s)}$.
8: Set $s = s + 1$.
9: **end while**

computationally expensive especially when the number of particles is high. Algorithm 5.1 describes our modified PF algorithm that supports multiscale subsampling and extension.

5.1 Particle Subsampling

In each cycle in Algorithm 5.1, we first resample a new set of k particles from the set $\mathcal{P}_n$ using their weights as the distribution function. Once we apply the dynamic model to each particle and advance it, new weights have to be computed. Therefore, we first select a small subset from all the n particles. The goal is to find a good set of representative particle candidates that will capture the geometry and the activity of the source weight function $w : \mathcal{P}_n \to \mathbb{R}$. To identify these candidates, we define a distance metric between the n particles using a weighted Euclidean distance between each two particles viewed as vectors. Other metrics can be used as well. We select the particle candidates using Algorithm 4.2 which is the randomized ID. We construct an affinity matrix $A^{(s)}$ that contains the affinities $d(p_i, p_j)$ between the particles. The kernel, which is defined between the particles, is

$$[A^{(s)}]_{ij} \triangleq \exp\left(\frac{-d(p_i, p_j)^2}{\epsilon_s}\right), i, j = 1, .., n. \qquad (5.1)$$

We calculate the affinities between all the particles in $\mathcal{P}_n$ such that $A^{(s)}$ is an $n \times n$ matrix defined by Eq. 5.1. The number of candidates we use is at most k. The output from the randomized ID algorithm (Algorithm 4.2) will be the set $\mathcal{P}_k$ of k particles that were selected from $\mathcal{P}_n$. We compute directly the weights for the k selected particles.

5.2 Weight Calculation using Function Extension

We obtained a set of particles $\mathcal{P}_k$ with their calculated weight values. Next, we continue and compute the weights for the rest of the particles that are not included in $\mathcal{P}_k$. We compute the weight value for each of the other $n - k$ particles by applying Algorithm 4.4 to the set $\mathcal{P}_k$ that has the first k columns of the affinity matrix $A^{(s)}$. These columns contain the affinities between each pair of particles in $\mathcal{P}_k$, the affinities between the particles in $\mathcal{P}_k$ and all the other particles. The output from Algorithm 4.4 is the weights of the $n - k$ particles that were not selected in the previous step. This extension method allows us to skip a direct weight computations for the remaining $n - k$ particles. Therefore, we keep the entire set of particles $\mathcal{P}_n$ from which we resample a small set of particles in the next PF algorithm step. This is especially beneficial when we cannot compute the weights for all particles if the computation is too expensive. Once the $n - k$ weights are calculated, we select the particle with the maximum likelihood as the prediction result and continue to the next algorithmic cycle.

Weights computation for k particles and their extension to the other $n - k$ particles reduce the number of operations and accelerates the PF as demonstrated in Section 7.

6 Accelerating the Particle Sampling Step

The computational bottleneck in Algorithm 5.1 lies in the selection step (Step 5) where we sample a subset of size k from the source particle set of size n. In this step, the randomized ID algorithm requires $\mathcal{O}(kn^2 + k^2 n \log n)$ operations (see Section 5.3 in [7]). In addition, the input kernel matrix calculation in the ID algorithm requires $\mathcal{O}(n^2)$ operations. Therefore, Algorithm 5.1 does not scale well and the gained performance boost decreases as the number of particles increases. To improve the performance of the sampling step, a different sampling method was developed. The method is based on a variation of the Farthest Point Sampling (FPS) algorithm [8], [31] combined with traditional kernel density estimation, which we refer to as Weighted-FPS (WFPS). The FPS algorithm begins by selecting a random data point and adding it into the sampled set. Then, in each step it adds the farthest data point from the sampled set, thus minimizing the distance between the original data points and the sampled set. The resulted sampled set contains k data points which spans the original set. This sampling step can be viewed as regular FPS selection with distance metric weighted by the particles densities to give priority

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

Algorithm 5.1: Multiscale PF (MSPF)
Input: n number of particles; x_0 initial state; $y_1, ..., y_T$ current observations; $q(\cdot)$ proposal distribution function; $p(\cdot)$ approximated posterior distribution function
Output: $x_1, ..., x_T$ estimated observations

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

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

5: Selection: Select a subset of size k from the new particles $\tilde{x}_t^{(i)}$ by computing the affinity matrix $A^{(s)}$ (Eq. 5.1) and by using the ID Algorithm 4.2.
6: Calculate the weights of the k selected particles using

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

7: Weight extension: Calculate the weights of the $n - k$ particles using the MSE Algorithm 4.4.
8: Weights normalization:

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

9: x_t is set to be the particle $\tilde{x}_t^{(l)}$ where $l = \arg\max_{1 \le i \le n} w_t^{(i)}$.
10: **end for**

to dense areas. The standard FPS algorithm is computed in $\mathcal{O}(k\log n)$ operations [32]. However, data points densities calculation, which uses a Gaussian kernel, takes $\mathcal{O}(n^2)$ operations. In our implementation, we use the multidimensional version of the Gauss Transform to calculate the density of each data point (in our case it is a particle). We use the Fast Multipole Method (FMM) as an efficient way to calculate the Gauss Transform that is called the Fast Gauss Transform (FGT). This approach enables to reduce the computational cost of the particle selection step from $\mathcal{O}(kn^2 + k^2 n\log n)$ to $\mathcal{O}(n + k\log n)$ operations. Sections 6.1 and 6.2 briefly describe the implementation of FMM and FGT, respectively. These descriptions were adopted from [33].

6.1 Fast Multipole Method

Assume that we want to evaluate the sum

$$v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j), j = 1, ..., M \quad (6.1)$$

where $\{\phi_i\}$ is a family of functions that correspond to a source function ϕ centered around different locations x_i, y_j is a point in a d-dimensional space and u_i is a weight. Direct evaluation of the sum in Eq. 6.1 requires $\mathcal{O}(MN)$ operations. In the FMM algorithm [9], we assume that $\{\phi_i\}$ can be expanded with a multipole series and with a local series centered at x_* and y_*, respectively, such that

$$\begin{aligned} \phi(y) &= \sum_{n=0}^{p-1} b_n(x_*) S_n(y - x_*) + \epsilon_S(p) \\ \phi(y) &= \sum_{n=0}^{p-1} a_n(y_*) R_n(y - y_*) + \epsilon_R(p) \end{aligned} \quad (6.2)$$

where S_n and R_n are the multipole and the local basis functions, respectively, x_* and y_* are the expansion centers, $\{a_n\}$ and $\{b_n\}$ are the expansion coefficients and $\epsilon_S(p)$ and $\epsilon_R(p)$ are the errors induced by truncating the series after p terms. Rewriting the sum in Eq. 6.1 using one of the expansions in 6.2 gives:

$$\begin{aligned} v(y_j) &= \sum_{i=1}^{N} u_i \phi_i(y_j) \\ &= \sum_{i=1}^{N} u_i \sum_{n=0}^{p-1} c_{ni} R_n(y_j - y_*), j = 1, ..., M. \end{aligned} \quad (6.3)$$

Here, c_{ni} is the coefficient a_n of ϕ_i. By rearranging the expression in Eq. 6.3 we get

$$\begin{aligned} v(y_j) &= \sum_{n=0}^{p-1} \left[\sum_{i=1}^{N} u_i c_{ni} \right] R_n(y_j - y_*) \\ &= \sum_{n=0}^{p-1} C_n R_n(y_j - y_*). \end{aligned} \quad (6.4)$$

The computation of Eq. 6.4 takes $\mathcal{O}(Mp + Np)$ operations where p determines the desired accuracy. The FMM can be used to compute the Gauss transform efficiently and this is referred as the FGT.

6.2 Fast Gauss Transform (FGT)

The FGT can be evaluated by using the FMM directly by choosing $\phi_i(y) = e^{-\|y - x_i\|^2/h^2}$, and then expanding the Gaussian using Hermite Polynomials. In one dimension, this yields

$$e^{-\|y - x_i\|^2/h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} \left(\frac{x_i - x_*}{h}\right)^n H_n\left(\frac{y - x_*}{h}\right) + \epsilon(p) \quad (6.5)$$

where H_n are Hermite polynomials defined by $H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}(e^{-x^2})$. Extension to higher dimensions is done by considering the multivariate Gaussian function as a product of univariate Gaussians where the series factorizations are applied to each dimension, see [33]. Equipped with a fast method to calculate the densities, we present in Section 6.3 the full Weighted Farthest Point Selection (WFPS) algorithm for selecting the representatives data points that replaces the randomized ID Algorithm 4.2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

6.3 Weighted Farthest Point Selection (WFPS) Algorithm

WFPS is a modification of the FPS Algorithm (described in Section 6). In WFPS, the metric value is bigger for two pairs of data points with equal distance that are located in a high density area than if they are located in an area with lower density. Therefore, the next sampled data point in each step in the FPS algorithm is selected according to a density-weighted distance function. The idea of replacing the ID algorithm with the WFPS is originated from the empirical observations that choosing the most linear independent columns of the affinity matrix by the ID is similar to choosing the actual data points based on distance and density. This can be viewed as selecting a data point which is the "most different" in distance terms. The density computation in the FPS algorithm is based on the observation that the ID algorithm favors columns of the affinity matrix that correspond to distant data points with high density. We found that the WFPS algorithm yields an improved particle selection set in comparison to either randomly selected or a regular FPS selection. The density weights cause the algorithm to prefer particle selections from dense areas. Similar augmentation to the FPS algorithm is shown in [34].

Algorithm 6.1 describes the WFPS algorithm for selecting k data points from a set of n data points in $\mathbb{R}^d$.

Algorithm 6.1: Weighted Farthest Point Selection (WFPS)
Input: A set of data points $X = \{x_1, .., x_n\}$ in $\mathbb{R}^d$; k the number selected data points
Output: k selected data points S

1: Set $w_1, ..., w_n$ to be the calculated densities of the data points in X using FGT (Eq. 6.4).
2: Set $S = \{x_1\}$.
3: Set $d_s(x_i) = w_i \|x_i - x_1\|$ for all data points in X.
4: **for** step=2,...,k **do**
5: Find the farthest data point in S:

$$s = \arg\max_{x \in X} d_s(x).$$

6: Add data point s to the set S.
7: Update the distances of the data points in X:

$$d_s(x_i) = min(d_s(x_i), w_i \|x_i - s\|).$$

8: **end for**

Intuitively, the WFPS selection method seems similar to the columns (particles) chosen by the ID when it is applied to the affinity matrix. The reason for using ID is to find a numerically stable basis of columns (i.e. matrix with a small condition number) to the affinity matrix. To verify that the suggested approach of WFPS provides a numerically stable columns (basis), we performed the following experiment: we generated a normally distributed set of N points ($N = 1000$), chose k points out of them using FPS with density estimation, built the $N \times k$ affinity matrix and computed its condition number. Then, we built the $N \times N$ affinity matrix and applied the ID to it (deterministic and random), chose k columns from it and computed the condition number. Next, we compared the obtained condition number to the one obtained using random sampling. Figs. 6.1 and 6.2 show that the condition number obtained using FPS with density estimation, is close to this obtained with ID but not as good. This provides a motivation for the WFPS as a faster approach and a motivation to the ID as a more accurate approach.

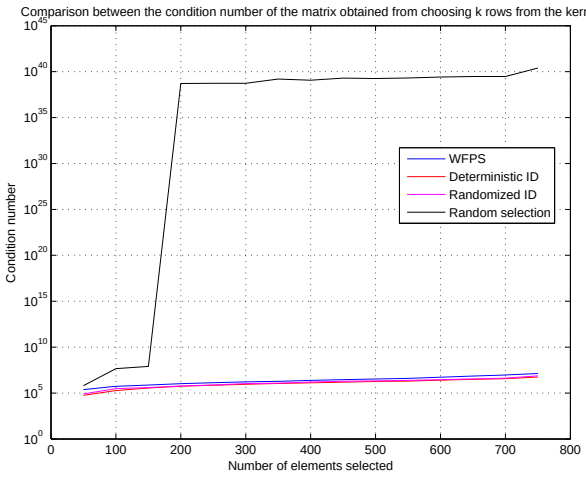


Fig. 6.1. Comparison between the condition number of a Gaussian kernel columns selected by different methods.

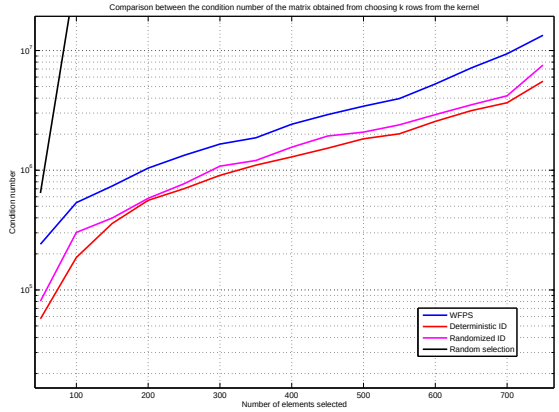


Fig. 6.2. Comparison between the condition number of a Gaussian kernel columns selected by different methods (zoomed).

When the selection step (Step 5) in the deterministic ID Algorithm 4.1 is replaced by the WFPS Algorithm 6.1 (where the input set to the WFPS is the particle set $\mathcal{P}_n$), we achieved even faster computational time. The results are presented in Section 7.4.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

7 Experimental Results

The performance of the MSPF algorithm was evaluated by preforming several experiments on objects tracking in both synthetic and real video sequences. The results were compared with the results from other tracking methods. We used a video sequence to track a ball (Fig. 7.1), which moves in a non-linear way around a basketball player. Each particle is described by a vector with six coordinates $p = (x, y, v_x, v_y, w, h)$, which are the target's location, speed in each axis, width (w) and height (h), respectively. The target's initial state p_0 is given as the input to the algorithm. Initially, the algorithm extracts a color histogram from a tile $\mathcal{B}_T$ that contains the target. The target's tile is a rectangular defined by four points

$$\begin{aligned}\mathcal{B} = \quad & \{(x - \tfrac{1}{2}w, y - \tfrac{1}{2}h), (x + \tfrac{1}{2}w, y - \tfrac{1}{2}h), \\ & (x - \tfrac{1}{2}w, y + \tfrac{1}{2}h), (x + \tfrac{1}{2}w, y + \tfrac{1}{2}h)\}. \end{aligned} \tag{7.1}$$

This tile is used later when it is compared to the other color histograms of the other particles. We also used the weighted Euclidean distance between two particles as the distance between the vectors that represents them such that

$$d(p^{(i)}, p^{(j)}) = \|p^{(i)} - p^{(j)}\|_2. \tag{7.2}$$

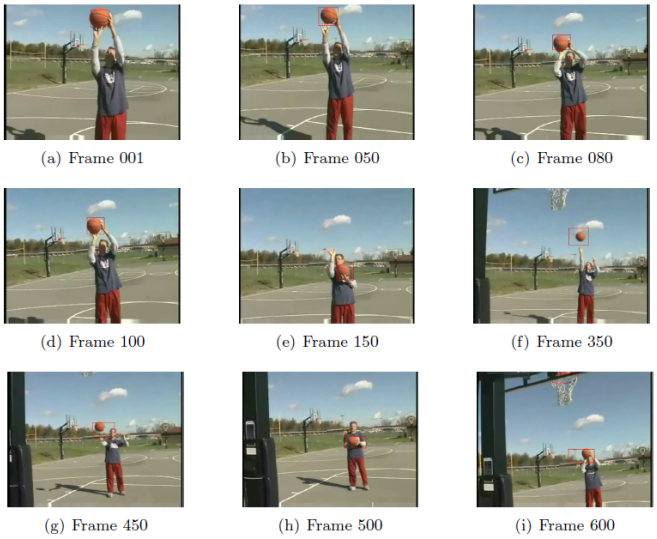
This metric was used in the affinity matrix $A^{(s)}$ (Eq. 4.3) in Algorithm 4.4. In each cycle of the algorithm, we sampled the particles and obtained a new set of n particles (see Step 3 in Algorithm 5.1). The model equations are applied to each particle. In this case, it is done by adding the speed to the corresponding location coordinates. Then, we perturbed each particle by adding a Gaussian noise with a standard deviation configured to each coordinate separately. The system dynamics is formulated as

$$p_t = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} p_{t-1} + n_t \tag{7.3}$$

where n_t is a random Gaussian noise vector such that $n_t(i) \sim \mathcal{N}(0, \sigma_i^2)$, and $\sigma_i^2, i = 1, \ldots, 6$, represents the variance we assigned for this coordinate. For example, in a constant velocity, the variance of the velocity is zero, that is $\sigma_{v_x}^2 = \sigma_{v_y}^2 = 0$. To calculate the weight of each particle $p^{(i)}, i = 1, \ldots, n$, we process a tile, which is centered at (x, y) with width w and height h defined in Eq. 7.1, by calculating a color histogram for all the pixels within the tile. Then, the histogram is compared with the histogram from the original target tile using the Bhattacharyya distance (Eq. 3.1). Therefore, the particle's weight is

$$w^{(i)} = \sqrt{\mathbf{h}(\mathcal{B}_T)^T \mathbf{h}(\mathcal{B}_i)}, i = 1, \ldots, n, \tag{7.4}$$

where $h(\mathcal{B}_i)$ is the color histogram of tile $\mathcal{B}_i$. In the next step, we compute the weights of the k particles selected by the randomized ID Algorithm 4.2. Their values are used for the weights calculation for the rest of the particles. This is done by applying the MSE with the defined distance metric (Eq. 7.2). Once all the weights were calculated, they are normalized and are used as the new distribution values for the n particles to be re-sampled in the next phase. The results from the application of the MSPF Algorithm 5.1 are applied to the basketball sequence and they are displayed in Fig. 7.1. The basketball is being tracked while the camera is moving and the background is changing constantly.



(a) Frame 001 (b) Frame 050 (c) Frame 080

(d) Frame 100 (e) Frame 150 (f) Frame 350

(g) Frame 450 (h) Frame 500 (i) Frame 600

Fig. 7.1. A set of representative frames from a basketball tracking sequence. The object is tracked using the MSPF Algorithm 5.1 with a direct computation of the weights for 10% from the total number of particles.

The performance of the MSPF Algorithm 5.1 was tested with different n and k values. We found that for most tracking tasks, a small set of particles (between 5% to 10% from the total number of particles n) is sufficient as seen in Fig. 7.1.

To measure the acceleration of the MSPF Algorithm 5.1, the tracking quality of each algorithm was tested along with their weight computation time. Each test was repeated 10 times and the average success tracking rate was computed. The standard PF (Algorithm 3.1) was tested with a particle range of $10-300$ particles. The MSPF Algorithm was tested using $100 - 3000$ particles while a direct computation was done for 10% of the particles. The results are shown in Fig. 7.2. We can see that in the same computation time, the tracking success ratio of MSPF Algorithm outperforms the standard PF (Algorithm 3.1).

In addition, the MSPF Algorithm 5.1 tracking success graph has less jitter than the standard PF Algorithm. This is due to the fact that the MSPF algorithm uses more particles to cover the same state space under similar computational cost then the standard PF Algorithm. For example, the standard PF execution time with 160

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

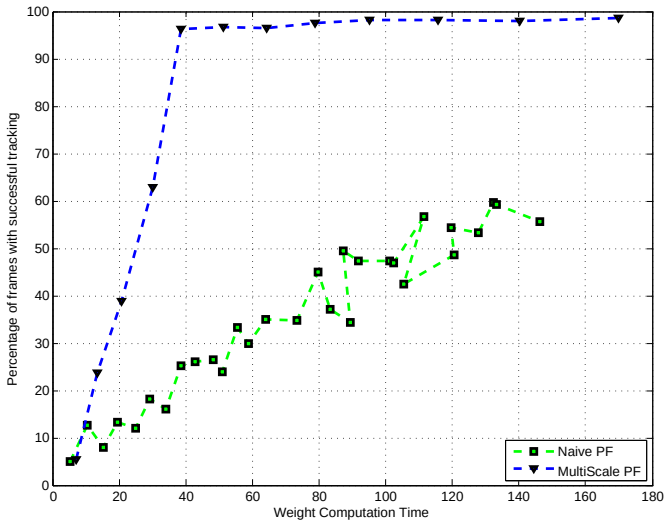


Fig. 7.2. Comparison between the tracking success rate for a given computational budget with standard PF (Alg. 3.1 which we refer to as the "naive PF") and MSPF (Alg. 5.1).

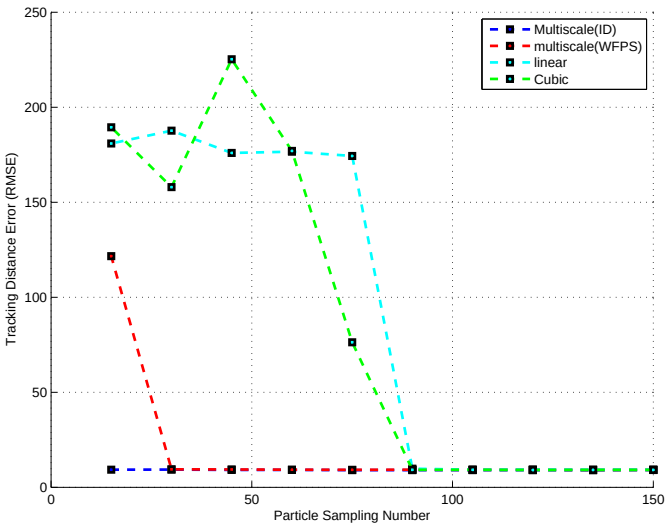


Fig. 7.3. Comparison between the RMSE for different methods: multiscale with ID sampling (Alg. 4.2), multiscale with WFPS sampling (Alg. 6.1), linear approximation and cubic approximation.

particles took 80 seconds while achieving a 45% tracking success rate. The MSPF achieved a 98% tracking success rate when using 800 particles where the weights were computed for only 10% of them while achieving the same execution time.

7.1 Comparison with Other Approximation Methods

In order to compare between the performance of the MSPF algorithm with different approximation methods, we tested the MSPF algorithm using different approximation methods to calculate the particle's weights. For 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 MSPF 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 MSPF algorithm output and the real location of the target. The MSPF Algorithm 5.1 achieved the lowest error rate even when we sampled between 2%-5% particles. When such subsampling rate was used, all the other tested methods fail (error grew).

Next, we compared between the computational time performing Algorithm 4.4 using different sampling rates, by running the PF Algorithm 3.1. The weights computation of all particles took 200 seconds (on average). From Fig. 7.4, we can see that the MSPF Algorithm 5.1 achieved the lowest computational time when the sampling rate was lower than 13% of the total number of particles. When the WFPS sampling was used, the computational time was even better. Overall, the MSPF

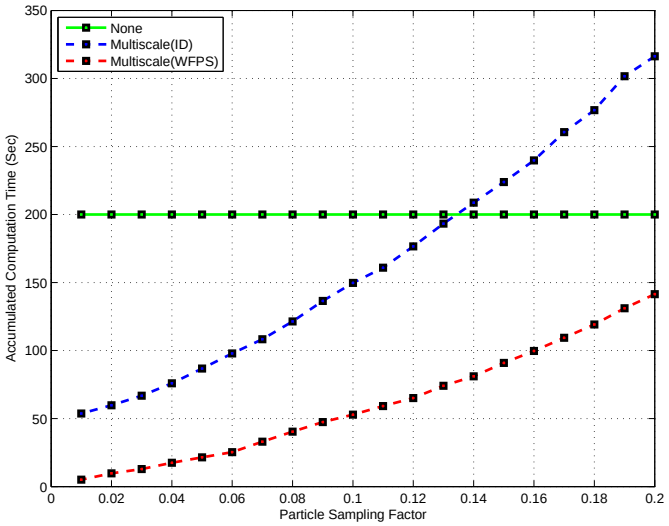


Fig. 7.4. Computational time of the MSPF with different sampling rates. The total number of particles is 1500.

Algorithm 5.1 achieved the lowest computational time while maintaining a low error rate.

We repeated the tests with another video sequence where the disc location was set to simulate Brownian motion such that the acceleration was a random white noise. The comparison between the running time and tracking error rate showed similar results as in Figs. 7.3 and 7.4.

7.2 Multiple Targets Tracking

The MSPF Algorithm 5.1 was tested on a video sequence that contains multiple objects. In such scenario, the tracking can be achieved by using two separate PF algorithms. Each PF uses a different set of particles and a separate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

set of observations. Here, 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 k 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 the other interpolation methods since it works better 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.

(a) Frame 001 (b) Frame 150 (c) Frame 250

(d) Frame 350 (e) Frame 400 (f) Frame 450

(g) Frame 500 (h) Frame 600 (i) Frame 700

Fig. 7.5. A selected set of representative frames from the tennis game that demonstrates the tracking performance. The two tennis players were tracked by the application of the MSPF Algorithm 5.1 with a direct weights computation for 10% from the total number of particles.

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 algorithmic 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 5.1, 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. Fig. 7.5 displays the results from the application of the MSPF Algorithm 5.1 to achieve multiple targets tracking. 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 Algorithm. The complete videos of the basketball and tennis games tracking can be viewed in our website[1].

7.3 Comparison with the EMD Measurement

Recently, the Earth Moving Distance (EMD) [35] was used for particles weight computation since this particle weight fits deformable objects [36]. The EMD computational cost is significantly higher than other methods such as color histograms. The MSPF becomes effective as the computational cost of the weights increases. We tested Algorithm 5.1 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. Each run used several frames executed on i7-2630QM 2.9GHz processor. Weights were calculated for 10% from the total number of particles while the rest of the particles were estimated using the MSE Algorithm. We verified that the target was not lost during the tracking procedure in each execution when using MSPF algorithm.

Table 7.1 shows the time differences between the standard version of the PF algorithm that uses the EMD metric (Algorithm 3.1) and our implementation that uses the MSE method (Algorithm 4.2) . For the latter, 10% of the particles were sampled, and the MSE was applied to the other 90% of the particles. We can see that the MSE algorithm reduces the PF total computation time. However, we can also observe that when the number of particles increases, the acceleration becomes less significant, as seen in the second and third columns. We analyze this scalability issue in Section 7.4.

7.4 Weighted FPS in the Selection Step

Table 7.1 compares between the running times of the WFPS Algorithm 6.1 and the randomized ID as the selection methods. Each time the MSPF Algorithm 5.1, which uses the EMD, was tested with a different particle set sizes. Performance comparisons were done between the following algorithms: standard PF, PF with MSE and ID selection, PF with MSE and WFPS selection. Table 7.1 shows that the acceleration factor is high even when $10,000$ particles are used.

1. http://www.cs.tau.ac.il/research/yaniv.shmueli/mspf/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 7.1
Comparison between WFPS and ID acceleration times [sec], in the MSPF algorithm that uses EMD. Sampling rate was 10% from the total number of particles.

# of Particles	Time [No MSE] Alg. 3.1	Time [MSE-ID] Alg. 4.2	Time [MSE-WFPS] Alg. 6.1	Acceleration Factor
2000	63	10.6	6.6	9.5
4000	125	32	14	8.9
6000	187	75.4	22	8.5
8000	260	151	32	8.1
10000	294	266	41	7.1

CONCLUSION

In this work, several contributions are presented. The PF computational time was reduced by the application of MSE method that reduces the load of the particle weight calculation. Therefore, it allows us to utilize more particles within a given computational budget. This improves the PF performance. The modified PF algorithm was tested on real video sequences to successfully track a single and multiple targets. In addition, the performance of the PF was compared with other extension methods and demonstrated that the MSE managed to track the target with much fewer particles then PF was using. These enhancements can become effective when multiple targets are tracked in real-time.

ACKNOWLEDGMENT

This research was partially supported by the Israel Science Foundation (Grant No. 1041/10), by the Israeli Ministry of Science & Technology (Grants No. 3-9096, 3-10898), by US - Israel Binational Science Foundation (BSF 2012282) and by a Fellowship from Jyväskylä University.

REFERENCES

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

[2] H. Cheng, Z. Gimbutas, P. Martinsson, and V. Rokhlin, "On the compression of low rank matrices," *SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1389–1404, 2005.

[3] A. Bermanis, A. Averbuch, and R. Coifman, "Multiscale data sampling and function extension," *Applied and Computational Harmonic Analysis*, vol. 34, pp. 15–29, 2013.

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

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

[6] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[7] P. Martinsson, V. Rokhlin, and M. Tygert, "A randomized algorithm for the decomposition of matrices," *Applied and Computational Harmonic Analysis*, vol. 30, no. 1, pp. 47–68, 2011.

[8] T. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[9] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 73, no. 2, pp. 325–348, 1987.

[10] N. Gupta, P. Mittal, S. Roy, S. Chaudhury, and S. Banerjee, "Developing a gesture-based interface," *Journal of the Institution of Electronics and Telecommunication Engineers*, vol. 48, no. 3, pp. 237–244, 2002.

[11] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "An adaptive color-based particle filter," *Image and Vision Computing*, vol. 21, no. 1, pp. 99–110, 2003.

[12] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1, pp. 99–141, 2001.

[13] X. Zhang, W. Hu, and S. Maybank, "A smarter particle filter," *Computer Vision–ACCV 2009*, pp. 236–246, 2010.

[14] M. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. of the American Statistical Association*, pp. 590–599, 1999.

[15] J. Kotecha and P. Djuric, "Gaussian sum particle filtering," *IEEE Transactions on Signal Processing*, vol. 51, no. 10, pp. 2602–2612, 2003.

[16] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan, "The unscented particle filter," *Advances in Neural Information Processing Systems*, pp. 584–590, 2001.

[17] M. Isard and A. Blake, "Condensationconditional density propagation for visual tracking," *International journal of computer vision*, vol. 29, no. 1, pp. 5–28, 1998.

[18] T.-J. Cham and J. M. Rehg, "A multiple hypothesis approach to figure tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2. IEEE, 1999.

[19] R. Urtasun, D. J. Fleet, and P. Fua, "3d people tracking with gaussian process dynamical models," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 238–245.

[20] K. Choo and D. J. Fleet, "People tracking using hybrid monte carlo filtering," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 321–328.

[21] C. Sminchisescu and B. Triggs, "Kinematic jump processes for monocular 3d human tracking," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1. IEEE, 2003, pp. I–69.

[22] J. Deutscher, A. Blake, and I. Reid, "Articulated body motion capture by annealed particle filtering," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2. IEEE, 2000, pp. 126–133.

[23] F. Wang and M. Lu, "Efficient visual tracking via hamiltonian monte carlo markov chain," *The Computer Journal*, 2012.

[24] E. Wan, A. Doucet, R. van der Merwe, and N. de Freitas, "The unscented particle filter," Technical report CUED/F-INFENG/TR380, Cambridge University, Tech. Rep., 2000.

[25] Y. Rui and Y. Chen, "Better proposal distributions: Object tracking using unscented particle filter," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 2. IEEE, 2001, pp. II–786.

[26] T. X. Han, H. Ning, and T. S. Huang, "Efficient nonparametric belief propagation with application to articulated body tracking," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 214–221.

[27] B. Han, Y. Zhu, D. Comaniciu, and L. S. Davis, "Visual tracking by continuous density propagation in sequential bayesian filtering framework," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 5, pp. 919–930, 2009.

[28] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House Publishers, 2004.

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

[30] G. Golub and C. Van Loan, *Matrix computations*. Johns Hopkins Univ Pr, 1996, vol. 3.

[31] A. Bronstein, M. Bronstein, and R. Kimmel, *Numerical geometry of non-rigid shapes*. Springer-Verlag New York Inc, 2008.

[32] T. Feder and D. Greene, "Optimal algorithms for approximate clustering," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 434–444.

[33] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis, "Improved fast gauss transform and efficient kernel density estimation," in *Ninth IEEE International Conference on Computer Vision, 2003.* IEEE, 2003, pp. 664–671.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2015.2392754, IEEE Transactions on Pattern Analysis and Machine Intelligence

[34] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.
[35] 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.
[36] 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.

Gil Shabat is a PhD Student in Tel Aviv University. Gil was born in Tel Aviv, Israel. He received his B.Sc and M.Sc degrees in Electrical Engineering from Tel-Aviv university, in 2002 and 2008, respectively. Prior to his PhD research, Gil was working in Applied Materials as a computer vision algorithm researcher. His research interests include low rank approximations, fast randomized algorithms, signal/image processing and machine learning.

Yaniv Shmueli is a Ph.D. candidate at the Computer Science School, Tel Aviv University. He received his B.Sc. and M.Sc. degrees in Mathematics and Computer Science from Tel Aviv University, Israel, in 1997 and 2003, respectively. Before starting his Ph.D. research, Yaniv worked at Kenshoo as R&D manager. Prior to that, Yaniv led the cable software group at Texas Instruments. His current research interests include dimensionality reduction, diffusion maps, machine learning, particle filters and matrix factorization methods.

Amit Bermanis received the B.Sc. degree in mathematics and computer science from the Technion-Israel Institute of Technology, Haifa, Israel, in 2003 and the M.Sc. and Ph.D. degrees in applied mathematics from Tel-Aviv University, Tel-Aviv, Israel, in 2007 and 2012, respectively. He is a postdoctoral fellow in the department of computer science, Tel Aviv University, Tel-Aviv, Israel. His research interests include methods for Big Data sampling and analysis.

Amir Averbuch was born in Tel Aviv, Israel. He received the B.Sc and M.Sc degrees in Mathematics from the Hebrew University in Jerusalem, Israel in 1971 and 1975, respectively. He received the Ph.D degree in Computer Science from Columbia University, New York, in 1983. During 1966-1970 and 1973-1976 he served in the Israeli Defense Forces. In 1976-1986 he was a Research Staff Member at IBM T.J. Watson Research Center, Yorktown Heights, in NY, Department of Computer Science. In 1987, he joined the School of Computer Science, Tel Aviv University, where he is now Professor of Computer Science. His research interests include applied harmonic analysis, big data processing and analysis, wavelets, signal/image processing, numerical computation and scientific computing (fast algorithms).