18th International Conference on Information Fusion
Washington, DC - July 6-9, 2015

Distributed Particle Filtering Via Optimal Fusion of Gaussian Mixtures

Jichuan Li and Arye Nehorai

The Preston M. Green Department of Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, MO, 63130 USA
Email: {lijichuan, nehorai}@ese.wustl.edu

***Abstract*—We propose a distributed particle filtering algorithm based on optimal fusion of local posterior estimates. We derive an optimal fusion rule from Bayesian statistics, and implement it in a distributed and iterative fashion via an average consensus algorithm. We approximate local posterior estimates as Gaussian mixtures, and fuse Gaussian mixtures through importance sampling. We prove that under certain conditions the proposed distributed particle filtering algorithm converges to a global posterior estimate locally available at every sensor in the network. Numerical examples are presented to demonstrate the performance advantages of the proposed method in comparison with other posterior-based distributed particle filtering algorithms.**

***Index Terms*—Distributed particle filtering, consensus, data fusion, Gaussian mixture model, importance sampling.**

I. Introduction

Particle filtering, also known as the sequential Monte Carlo method, is a powerful tool in sequential Bayesian state estimation [1]. Unlike Kalman filtering [2], particle filtering works with nonlinear models and non-Gaussian noises, and is thus applicable to sequential estimation problems under more general assumptions. To improve estimation accuracy, a particle filter is often built on observations from more than one perspective or sensor. These sensors form a network and collaborate via communication. The network can designate one of the sensors as a fusion center that receives and processes observations sent from all the other sensors in the network. This centralized implementation provides optimal accuracy, but does not scale with growing network size in applications like target tracking, environment monitoring, and smart grids, which motivates distributed particle filtering [3].

Distributed particle filtering consists of separate particle filters that have access to local measurements only, but produce global estimates based on information exchanged among themselves. It is often implemented using a consensus algorithm [4], where sensors in a network reach agreement in their beliefs iteratively through local communications between neighboring sensors. Depending on the information communicated in the consensus algorithm, a distributed particle filtering algorithm can be categorized as weight-based, likelihood-based, or posterior-based.

Weight-based algorithms [5], [6] communicate the weight of each particle in an ensemble. In order to approximate a distribution accurately, an ensemble has to contain a sufficiently large number of particles, which results in a considerably large amount of information to be communicated in a weight-based algorithm. Also, in order for weight consensus to make sense, different filters must have ensembles of identical particles, which requires perfect synchronization between the random number generators of different sensors. Although synchronization methods have been well developed for sensor networks [7]-[9], the dependence on perfect synchronization, together with the high communication overhead, makes weight-based algorithms costly to implement in practice.

Likelihood-based algorithms [10]-[12] communicate local likelihood functions. The fusion of local likelihood functions is straightforward if sensor observations are uncorrelated, but the communication is challenging. In order to be communicated in a compact form, a likelihood function has to be parametrically represented. Since a likelihood function varies greatly with both the observation model and the observation noise, there are few, if any, parametric approaches that apply to an arbitrary likelihood function in a sufficiently accurate way, not to mention that the parametric representation has to be easy to fuse. Hence, likelihood consensus might not be an ideal choice for general applications.

Posterior-based algorithms [13]-[16] communicate local posterior estimates of states. Unlike likelihood functions, posterior estimates are essentially density functions, and thus easy to represent parametrically. If an estimate follows a (multivariate) Gaussian distribution, it can be losslessly represented by its mean and variance (covariance matrix); if an estimate follows a non-Gaussian distribution, it can be sufficiently accurately approximated by a convex combination of multiple Gaussian components, i.e., a Gaussian mixture (GM) [17]. However, local posterior estimates are not always easy to fuse. In [13] and [14], local posterior estimates are fused in a Bayesian fashion, but assumed to be Gaussian for convenience of fusion. As we know, a posterior estimate follows a Gaussian distribution only if both the state transition model and the observation model are linear with additive Gaussian noises. Thus, the Gaussian assumption is generally too strong, and will incur obvious approximation errors in many applications. In [15] and [16], local posterior estimates are approximated with Gaussian mixtures, but fused linearly. The linear fusion rule is, however, suboptimal, because it is not supported by

underlying statistical theories.

In this paper, we propose a distributed particle filtering algorithm based on posterior estimates. We use Gaussian mixtures to accurately model posterior estimates, and fuse local estimates via an optimal distributed fusion rule derived from Bayesian statistics and implemented through consensus. Unlike other posterior-based algorithms, the proposed algorithm seeks consensus on the posterior distribution, rather than on parameters of the posterior distribution, thus giving flexibility to local approximations and allowing each Gaussian mixture to use an optimal but often nonuniform number of components. We design algorithms to fuse Gaussian mixtures within each consensus step. Finally, we prove the convergence of the proposed distributed particle filtering algorithm and demonstrate its advantages through numerical examples.

The rest of the paper is organized as follows. Section II introduces the sensor network model and the state-space model. Section III introduces centralized particle filtering. Section IV presents our distributed particle filtering algorithm. Section V proves the convergence of the proposed algorithm. Section VI presents numerical examples. Section VII concludes the paper.

II. Problem Formulation

A. Network Model

We model a sensor network as a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V} = \{S_1, S_2, \ldots, S_K\}$ is the set of vertices or sensors with cardinality $|\boldsymbol{V}| = K$, and $\boldsymbol{E}$ is the set of edges or communication links between sensors. We assume each communication link to be bidirectional, and thus the graph $\boldsymbol{G}$ to be undirected. We restrict each communication link to a local neighborhood, in the sense that a sensor can directly communicate only with its neighbors. Also, we assume that the graph $\boldsymbol{G}$ is connected, or in other words that there exists a multi-hop communication route connecting any pair of sensors in the network. In addition, we assume the sensor network to be synchronized.

B. Signal Model

We connect target state transition with sensor observation using a discrete-time state-space model,

$$\begin{cases} \boldsymbol{x}_n = \boldsymbol{g}(\boldsymbol{x}_{n-1}) + \boldsymbol{u}_n \\ \boldsymbol{y}_{n,k} = \boldsymbol{h}_k(\boldsymbol{x}_n) + \boldsymbol{v}_{n,k} \quad (k = 1, 2, \ldots, K) \end{cases}, \tag{1}$$

where

1) $\boldsymbol{x}_n \in \mathbb{R}^d$ is the target state at the nth moment;
2) $\boldsymbol{y}_{n,k} \in \mathbb{R}^b$ is the observation taken by S_k at the nth moment;
3) $\boldsymbol{g}$ is a known state transition function;
4) $\boldsymbol{h}_k$ is a known observation function of S_k;
5) $\{\boldsymbol{u}_n\}$ and $\{\boldsymbol{v}_{n,k}\}$ are uncorrelated additive noise;
6) the distribution of $\boldsymbol{x}_0$ is given as prior information;
7) state transition is Markovian, i.e., past and future states are conditionally independent, given the current state;
8) the current observation is conditionally independent of past states and observations, given the current state.

C. Goal

The goal is to sequentially estimate the current state $\boldsymbol{x}_n$ based on the currently available observations $\boldsymbol{y}_{1:n}$.

D. Notation

We represent consecutive states $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ as $\boldsymbol{x}_{1:n}$, consecutive observations $\{\boldsymbol{y}_{1,k}, \boldsymbol{y}_{2,k}, \ldots, \boldsymbol{y}_{n,k}\}$ taken by S_k as $\boldsymbol{y}_{1:n,k}$, observations taken by the network at the nth moment $\{\boldsymbol{y}_{n,1}, \boldsymbol{y}_{n,2}, \ldots, \boldsymbol{y}_{n,K}\}$ as $\boldsymbol{y}_n$, and consecutive observations $\{\boldsymbol{y}_{1,1}, \boldsymbol{y}_{1,2}, \ldots, \boldsymbol{y}_{n,K}\}$ by the network as $\boldsymbol{y}_{1:n}$. We use f to denote a probability density function (pdf), and q to denote the pdf of a proposal distribution.

III. Centralized Particle Filtering

The problem formulated in Section II is a filtering problem. A filtering problem is often solved by a particle filter when the state-space model is nonlinear or the noises are non-Gaussian. A particle filter can be implemented in a centralized fashion by collecting observations from all the sensors in the network and processing them together.

A centralized particle filter approximates the posterior distribution of the current state, $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$, as a weighted ensemble of Monte Carlo samples, also known as particles:

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \approx \sum_{m=1}^{M} w_n^{(m)} \delta(\boldsymbol{x}_n - \boldsymbol{x}_n^{(m)}), \tag{2}$$

where M is the number of particles, $\boldsymbol{x}_n^{(m)}$ is the mth particle, $w_n^{(m)}$ is the weight of $\boldsymbol{x}_n^{(m)}$ with $\sum_{m=1}^{M} w_n^{(m)} = 1$, and δ is the Dirac delta function. Using importance sampling [18], a particle is generated according to a proposal distribution $q(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{1:n-1}^{(m)}, \boldsymbol{y}_n)$, and the weight is updated according to

$$w_n^{(m)} \propto \frac{f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)}) f(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{n-1}^{(m)})}{q(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{1:n-1}^{(m)}, \boldsymbol{y}_n)} \times w_{n-1}^{(m)}. \tag{3}$$

The proposal distribution q is commonly chosen as the state transition pdf $f(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{n-1}^{(m)})$, which, although suboptimal, yields a convenient weight update rule:

$$w_n^{(m)} \propto f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)}) \times w_{n-1}^{(m)}. \tag{4}$$

The likelihood function $f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)})$ is global, and can be factorized into the product of local likelihood functions, $\prod_{k=1}^{K} f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n^{(m)})$, thus providing a centralized fusion rule.

Due to the finite number of particles, the weight in an ensemble tends to be concentrated in only a few particles as time goes on, resulting in a small effective sample size and thus a poor approximation. When an ensemble's effective sample size is lower than a threshold, a remedy is to resample the ensemble according to the particle weights. A frequently used estimate of the effective sample size is

$$\hat{M}_e = \frac{1}{\sum_{m=1}^{M} (w_n^{(m)})^2}, \tag{5}$$

and the threshold can be set as, for example, 60% of the original sample size M, or 100% if one plans to resample in every iteration.

Although centralized particle filtering is optimal, it is impractical for large-scale sensor networks. First, it expends considerable energy and bandwidth on transmitting raw measurements from everywhere in the network to a common fusion center. Also, its dependence on a single fusion center makes it vulnerable to single point failure. Moreover, it does not scale with the network size. Therefore, it is often preferable to perform distributed particle filtering.

IV. Distributed Particle Filtering

In distributed particle filtering, every sensor in the network performs separate particle filtering on its own observations and communicates locally with its neighbors about their posterior estimates for the purpose of data fusion.

A. Consensus

Consensus [4] is a type of data fusion algorithm in which each agent in the network iteratively communicates with its neighbors and updates its own belief in a certain fact based on its neighbors' beliefs, until convergence. We fuse local posterior estimates from different sensors via consensus, so that every sensor in the network ultimately obtains the same global posterior estimate.

Likelihood factorization, as mentioned in Section III, makes data fusion convenient, because its logarithmic form

$$\log f(\boldsymbol{y}_n|\boldsymbol{x}_n) = \sum_{k=1}^{K} \log f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) \tag{6}$$

gives rise to a straightforward implementation of an average consensus algorithms [4]. However, unlike a prior or posterior density function, a likelihood function is difficult to approximate parametrically through a generally applicable approach like the Gaussian mixture model. The consequent inconvenience in communicating likelihood functions motivates us to communicate posterior density functions instead.

Due to conditional independence, a likelihood function can be equivalently written as

$$f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) = f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n, \boldsymbol{y}_{1:n-1}), \tag{7}$$

which, according to Bayes' Theorem, can be rewritten as

$$f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) = \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})}. \tag{8}$$

Substitute (8) into the original form of (6), and then we get

$$\begin{aligned} &\log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_n, \boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} \\ =&\sum_{k=1}^{K} \log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})}, \end{aligned} \tag{9}$$

which yields

$$\log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_n, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} = \sum_{k=1}^{K} \log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} + const. \tag{10}$$

The constant term comes from the density functions $f(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})$ and $f(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})$, because they do not involve state variables.

(10) presents a centralized fusion rule for local posterior estimates. $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$ on the right-hand side of (10) is a local posterior estimate obtained by S_k based on its current observation $\boldsymbol{y}_{n,k}$ and the network's past observations $\boldsymbol{y}_{1:n-1}$, while $f(\boldsymbol{x}_n|\boldsymbol{y}_n, \boldsymbol{y}_{1:n-1})$ on the left-hand side of (10) is a global posterior estimate based on the network's current and past observations $\boldsymbol{y}_{1:n}$. There are two other terms in (10), namely $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$ and the constant term, but they do not affect data fusion. $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$, the prediction of the current state based on the network's past observations, can be calculated from earlier estimates as

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \int f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1}) \mathrm{d}\boldsymbol{x}_{n-1}.$$

The constant term will disappear when we normalize a posterior density function so that it integrates to 1.

Similar to (6), (10) can be iteratively implemented using an average consensus algorithm as

$$\begin{aligned} \log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i+1)}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} &= const + \log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} \\ + \epsilon \sum_{j \in N_k} \left(\log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,j}^{(i)}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} - \log \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} \right). \end{aligned} \tag{11}$$

Here $\boldsymbol{y}_{n,k}^{(i)}$ denotes the weighted set of observations fused into the belief of S_k up to the ith consensus iteration during the nth iteration of particle filtering, N_k denotes the index set of neighbors of S_k (with S_k excluded), and $\epsilon \in (0, 1/\max_k |N_k|)$ is a parameter controlling the consensus step size. (11) can be simplified to

$$\begin{aligned} \log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i+1)}, \boldsymbol{y}_{1:n-1}) = \epsilon \sum_{j \in N_k} &\log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,j}^{(i)}, \boldsymbol{y}_{1:n-1}) \\ + (1 - \epsilon|N_k|) &\log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1}) + const, \end{aligned} \tag{12}$$

which we call the distributed fusion step. Each sensor updates its own belief in the posterior distribution of $\boldsymbol{x}_n$ according to (12) in each consensus iteration, and disseminates its updated belief to its neighbors in the next consensus iteration. According to the convergence property of average consensus [4], under certain conditions, for $\forall\ k$, we have

$$\lim_{i \to \infty} \log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1}) = \frac{1}{K} \sum_{j=1}^{K} \log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,j}^{(0)}, \boldsymbol{y}_{1:n-1}), \tag{13}$$

where $\boldsymbol{y}_{n,j}^{(0)} = \boldsymbol{y}_{n,j}$. With (10) and (13), we have

$$f(\boldsymbol{x}_n|\boldsymbol{y}_n, \boldsymbol{y}_{1:n-1}) \propto \frac{\left(\lim_{i \to \infty} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1}) \right)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}}, \tag{14}$$

which, called the recovery step, concludes the consensus.

B. Gaussian Mixture Model

Consensus necessitates inter-sensor communication. Communication is a major source of energy consumption for most wireless sensor networks. Since wireless sensor networks are usually subject to strong energy constraints, it is important to minimize the amount of communication needed in consensus. A possible solution to communication minimization is to compress the data to be transmitted. In our problem, the data to be transmitted are posterior density functions, and we compress them using a parametric approach, the Gaussian mixture model [17]. A Gaussian mixture is a convex combination of Gaussian distribution components,

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n,k},\boldsymbol{y}_{1:n-1}) = \sum_{c=1}^{C} \alpha_c \, \mathcal{N}\left(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right), \qquad (15)$$

where C is the number of components, and α_c, $\boldsymbol{\mu}_c$, and $\boldsymbol{\Sigma}_c$ are the weight, mean, and covariance matrix of the cth Gaussian component, respectively.

A Gaussian mixture model can be used to approximate an arbitrary probability distribution, and is often learned via the expectation-maximization (EM) algorithm [19] from samples generated from the underlying distribution. Most Gaussian mixture learning algorithms assume the samples to be unweighted. However, samples or particles in particle filtering are usually weighted. Admittedly, resampling a weighted ensemble can produce an unweighted ensemble, from which a Gaussian mixture model can be conveniently learned, but it incurs information loss, since resampling provides only a discretized and stochastic approximation of the original ensemble. To avoid unnecessary information loss, it is always preferable, if possible, to learn a Gaussian mixture directly from weighted samples. Based on the EM approach on unweighted samples, we propose Algorithm 1 for Gaussian mixture learning on weighted samples.

Algorithm 1 GM learning on weighted samples

procedure GMLEARN($\{\boldsymbol{x}_i, w_i\}_{i=1}^N$, C)
  initialize C (if not given) and $\{\alpha_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^C$
  repeat
    for $i = 1$ to N **do** ▷ E-step
      for $c = 1$ to C **do**
        $p_{ic} = \alpha_c \, \mathcal{N}\left(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right)$
      end for
      normalize $\{p_{ic}\}_{c=1}^C$
    end for
    for $c = 1$ to C **do** ▷ M-step
      $\alpha_c = \sum_{i=1}^N p_{ic} w_i$
      $\boldsymbol{\mu}_c = \frac{1}{\alpha_c} \sum_{i=1}^N p_{ic} w_i \boldsymbol{x}_i$
      $\boldsymbol{\Sigma}_c = \frac{1}{\alpha_c} \sum_{i=1}^N p_{ic} w_i (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$
    end for
    normalize $\{\alpha_c\}_{c=1}^C$
  until convergence
  return GM $= \{\alpha_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^C$
end procedure

C. Fusion of Gaussian Mixtures

For posterior distributions approximated by Gaussian mixtures, the fusion of Gaussian mixtures has to be considered for both the distributed fusion step (12) and the final recovery step (14).

For convenience, we convert (12) from the current logarithmic form to its original form,

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i+1)}, \boldsymbol{y}_{1:n-1}) \propto f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})^{1-\epsilon|N_k|} \times \prod_{j\in N_k} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,j}^{(i)}, \boldsymbol{y}_{1:n-1})^{\epsilon}, \qquad (16)$$

which, similar to (14), involves products of powers of Gaussian mixtures. Unfortunately, the fractional exponents in (16) and the negative exponents in (14) make analytical computation intractable, thus motivating us to consider sampling. Since it is more efficient to sample from a distribution known to be close to the distribution to be found, than to sample from a generic distribution, e.g., the uniform distribution, we propose to fuse Gaussian mixtures in both (16) and (14) via importance sampling.

For (16), we generate samples $\{\boldsymbol{x}_n^{(m)}\}_{m=1}^M$ from a proposal distribution, $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})$, and assign them uniform weights. Then, the fusion step (16) becomes equivalent to updating weights $\{w_n^{(m)}\}_{m=1}^M$ according to

$$w_n^{(m)*} \propto w_n^{(m)} \times \prod_{j\in N_k} f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{n,j}^{(i)}, \boldsymbol{y}_{1:n-1})^{\epsilon} \times \frac{f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})^{1-\epsilon|N_k|}}{f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})}, \qquad (17)$$

where the asterisk in $w_n^{(m)*}$ indicates that the weight has been updated with the neighbors' information, and the division by the proposal distribution is a result from importance sampling. (17) can be simplified to

$$w_n^{(m)*} \propto \prod_{j\in N_k} f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{n,j}^{(i)}, \boldsymbol{y}_{1:n-1})^{\epsilon} \times f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})^{-\epsilon|N_k|}. \qquad (18)$$

After normalizing $\{w_n^{(m)*}\}_{m=1}^M$, a Gaussian mixture can be learned from the weighted ensemble $\{\boldsymbol{x}_n^{(m)}, w_n^{(m)*}\}_{m=1}^M$ using Algorithm 1. The fusion step is summarized in Algorithm 2.

Algorithm 2 GM fusion

procedure GMFUSE(GM$_k$, $\{\text{GM}_j\}_{j\in N_k}$)
  initialize M, ϵ
  generate $\{\boldsymbol{x}^{(m)}\}_{m=1}^M$ from GM$_k$
  for $m = 1$ to M **do**
    $w^{(m)} = f(\boldsymbol{x}^{(m)}|\text{GM}_k)^{-\epsilon|N_k|} \prod_{j\in N_k} f(\boldsymbol{x}^{(m)}|\text{GM}_j)^{\epsilon}$
  end for
  normalize $\{w^{(m)}\}_{m=1}^M$
  return GMLEARN($\{\boldsymbol{x}^{(m)}, w^{(m)}\}_{m=1}^M$)
end procedure

The recovery step (14) can be similarly implemented through importance sampling, as shown in Algorithm 3, where $\text{GM}_{\text{p}k}$ denotes the Gaussian mixture representation of the prediction of the current state by S_k based on the network's past observations.

Algorithm 3 GM recovery

procedure GMRECOVER(GM_k, $\text{GM}_{\text{p}k}$)
  initialize M
  generate $\{\boldsymbol{x}^{(m)}\}_{m=1}^{M}$ from $\text{GM}_{\text{p}k}$
  for $m = 1$ to M **do**
    $w^{(m)} = \left[f(\boldsymbol{x}^{(m)}|\text{GM}_k)/f(\boldsymbol{x}^{(m)}|\text{GM}_{\text{p}k})\right]^K$
  end for
  normalize $\{w^{(m)}\}_{m=1}^{M}$
  return GMLEARN($\{\boldsymbol{x}^{(m)}, w^{(m)}\}_{m=1}^{M}$)
end procedure

D. Summary

We summarize the proposed consensus-based distributed particle filtering algorithm in Algorithm 4.

Algorithm 4 Distributed particle filtering

procedure DPF($\{\boldsymbol{x}_{n-1,k}^{(m)}, w_{n-1,k}^{(m)}\}_{m=1,k=1}^{M,K}$, $\{\boldsymbol{y}_k\}_{k=1}^{K}$)
  for $k = 1$ to K **do** ▷ filtering
    for $m = 1$ to M **do**
      generate $\boldsymbol{u}_{n,k}^{(m)}$ from $f(\boldsymbol{u}_n)$
      $\boldsymbol{x}_{n,k}^{(m)} = \boldsymbol{g}(\boldsymbol{x}_{n-1,k}^{(m)}) + \boldsymbol{u}_{n,k}^{(m)}$
      $w_{n,k}^{(m)} = w_{n-1,k}^{(m)} \times f(\boldsymbol{y}_k|\boldsymbol{x}_{n,k}^{(m)})$
    end for
    normalize $\{w_{n,k}^{(m)}\}_{m=1}^{M}$
    $\text{GM}_{\text{p}k}$ = GMLEARN($\{\boldsymbol{x}_{n,k}^{(m)}, w_{n-1,k}^{(m)}\}_{m=1}^{M}$)
    GM_k = GMLEARN($\{\boldsymbol{x}_{n,k}^{(m)}, w_{n,k}^{(m)}\}_{m=1}^{M}$)
  end for
  repeat ▷ fusion
    for $k = 1$ to K **do**
      GM_k = GMFUSE(GM_k, $\{\text{GM}_j\}_{j\in N_k}$)
    end for
  until convergence
  for $k = 1$ to K **do** ▷ recovery
    GM_k^* = GMRECOVER(GM_k, $\text{GM}_{\text{p}k}$)
    generate $\{\boldsymbol{x}_{n,k}^{(m)}\}_{m=1}^{M}$ from GM_k^*
  end for
  return $\{\boldsymbol{x}_{n,k}^{(m)}, 1/M\}_{m=1,k=1}^{M,K}$
end procedure

V. CONVERGENCE

The convergence of an average consensus algorithm has been proved in [4]. The proof, however, cannot be directly applied to our problem, because our problem has a different formulation. The distributed fusion step (12) involves an additional constant term, which, together with the use of Gaussian mixture models, necessitates an additional normalization step at the end of each fusion step. Also, the centralized fusion equation (10) involves another density function of $\boldsymbol{x}_n$, $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$, in addition to $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$, which we seek consensus on. Although $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$ disappears in the distributed fusion step (12), it comes back in the recovery step (14). In summary, our problem poses a more complicated formulation than an ordinary average consensus algorithm can address. For rigorousness, we show the convergence of our method in this section.

A. Proof of Convergence

Theorem 1. *After iterations of distributed fusion (12) followed by recovery (14), the estimate held by each sensor converges to* $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$.

Proof. First, we study intermediate results from distributed fusion. For simplicity of notation, we denote $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})$ as $z_k^{(i)}$. Then, the exponential form of (12) can be written as

$$z_k^{(i+1)} = e^{const}\left(z_k^{(i)}\right)^{1-\epsilon|N_k|}\prod_{j\in N_k}\left(z_j^{(i)}\right)^{\epsilon}, \tag{19}$$

where e^{const} is a constant coefficient. Each consensus iteration involves a constant coefficient, and the constant coefficient accumulates across iterations. We denote the part of $z_k^{(i)}$ coming purely from the fusion of the original estimates from separate particle filters (or $z_k^{(0)}$ in other words) as $p_k^{(i)}$, and the accumulated constant coefficient that $z_k^{(i)}$ collects up to the ith iteration as $\lambda_k^{(i)}$. Then, we have $z_k^{(i)} = \lambda_k^{(i)} p_k^{(i)}$. When $i = 0$, we have $\lambda_k^{(0)} = 1$ and $p_k^{(0)} = f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$. When $i \geq 1$, we have

$$\begin{aligned} z_k^{(i+1)} =& e^{const}\left(z_k^{(i)}\right)^{1-\epsilon|N_k|}\prod_{j\in N_k}\left(z_j^{(i)}\right)^{\epsilon} \\ =& e^{const}\left(\lambda_k^{(i)} p_k^{(i)}\right)^{1-\epsilon|N_k|}\prod_{j\in N_k}\left(\lambda_j^{(i)} p_j^{(i)}\right)^{\epsilon} \\ =& \underbrace{e^{const}\left(\lambda_k^{(i)}\right)^{1-\epsilon|N_k|}\prod_{j\in N_k}\left(\lambda_j^{(i)}\right)^{\epsilon}}_{\lambda_k^{(i+1)}} \\ & \times \underbrace{\left(p_k^{(i)}\right)^{1-\epsilon|N_k|}\prod_{j\in N_k}\left(p_j^{(i)}\right)^{\epsilon}}_{p_k^{(i+1)}}. \end{aligned} \tag{20}$$

The logarithmic form of the last term $p_k^{(i+1)}$ is

$$\log p_k^{(i+1)} = (1-\epsilon|N_k|)\log p_k^{(i)} + \epsilon\sum_{j\in N_k}\log p_j^{(i)}, \tag{21}$$

which coincides with the canonical form of an average consensus algorithm. With the underlying graph $\boldsymbol{G}$ being connected and balanced (because it is undirected), according to [4] we have

$$\lim_{i\to\infty}\log p_k^{(i)} = \frac{1}{K}\sum_{k=1}^{K}\log p_k^{(0)}, \tag{22}$$

or equivalently

$$\lim_{i\to\infty} p_k^{(i)} = \prod_{k=1}^{K} \left(p_k^{(0)}\right)^{\frac{1}{K}} = \prod_{k=1}^{K} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})^{\frac{1}{K}}. \quad (23)$$

Hence,

$$\begin{aligned}\lim_{i\to\infty} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1}) &= \lim_{i\to\infty} z_k^{(i)} \\ &= \lim_{i\to\infty} \lambda_k^{(i)} p_k^{(i)} = \lim_{i\to\infty} \lambda_k^{(i)} \lim_{i\to\infty} p_k^{(i)} \\ &= \left(\lim_{i\to\infty} \lambda_k^{(i)}\right) \prod_{k=1}^{K} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})^{\frac{1}{K}}. \quad (24)\end{aligned}$$

Based on (24), the intermediate result from distribution fusion, we show that the final result from recovery converges to the desired posterior distribution $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$.

The right-hand side of the recovery equation (14) can be rewritten as

$$\begin{aligned}&\frac{\left(\lim_{i\to\infty} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}^{(i)}, \boldsymbol{y}_{1:n-1})\right)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}} \\ &= \left(\lim_{i\to\infty} \lambda_k^{(i)}\right)^K \frac{\prod_{k=1}^{K} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}}, \quad (25)\end{aligned}$$

where $\left(\lim_{i\to\infty} \lambda_k^{(i)}\right)^K$ is a constant, and

$$\begin{aligned}&\frac{\prod_{k=1}^{K} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}} \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) \prod_{k=1}^{K} \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})} \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) \prod_{k=1}^{K} \frac{f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n, \boldsymbol{y}_{1:n-1})}{f(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})} \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) \prod_{k=1}^{K} f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) \times const \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_n|\boldsymbol{x}_n) \times const \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_n, \boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1}) \times const \\ =&f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \times const. \quad (26)\end{aligned}$$

Therefore, normalizing the right-hand side of (14) gives each S_k the global posterior estimate $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$. □

B. Errors in Convergence

The convergence shown in Section V-A is built on the basis of three asymptotic assumptions: (i) the number of consensus iterations is sufficiently large, (ii) the number of generated samples is sufficiently large, and (iii) the approximation error of a Gaussian mixture model is sufficiently small. In practice, however, none of them can be perfectly satisfied without a tremendous amount of communication and computation. Hence, convergence errors always exist, although often small. Errors in convergence result in errors in consensus. To manually eliminate the consensus errors, we can run an additional average consensus algorithm on parameters of Gaussian mixtures. To do so, we have to guarantee that the number of Gaussian mixture components is in agreement among different sensors in the network, by, for example, re-learning from samples the Gaussian mixture models with a specified number of components. As mentioned in the Introduction, average consensus on parameters of Gaussian mixtures is suboptimal, but the suboptimality is insignificant here, because it is simply used to numerically fine-tune estimates that are already considerably close to consensus.

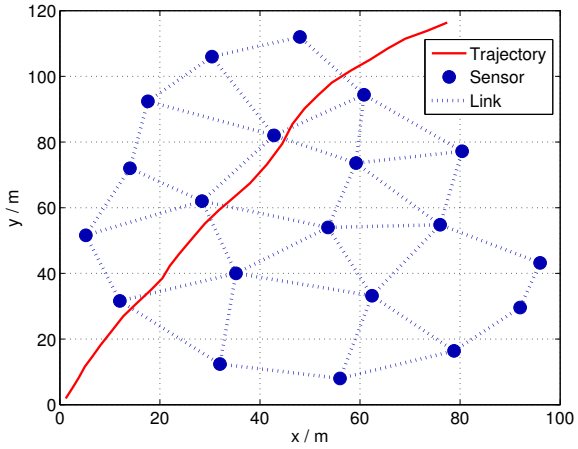

Figure 1. A wireless sensor network with local communication links, and a target trajectory to be estimated.

VI. Numerical Examples

In this section, we use numerical examples of target tracking to demonstrate the performance of the proposed distributed particle filtering algorithm in comparison with other posterior-based algorithms.

We tested these algorithms on a wireless sensor network consisting of 20 sensors that tried to track a moving target, as shown in Fig. 1. The target followed a noisy constant velocity kinematic model in a 2-dimensional space, with the state transition function

$$\boldsymbol{g}(\boldsymbol{x}_n) = \boldsymbol{D} \cdot \boldsymbol{x}_n, \quad (27)$$

where

$$\boldsymbol{x}_n = \begin{bmatrix} x_{n,1} \\ x_{n,2} \\ \dot{x}_{n,1} \\ \dot{x}_{n,2} \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (28)$$

and the transition noise

$$\boldsymbol{u}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}), \quad (29)$$

where

$$\boldsymbol{R} = \sigma_u^2 \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{bmatrix}. \quad (30)$$

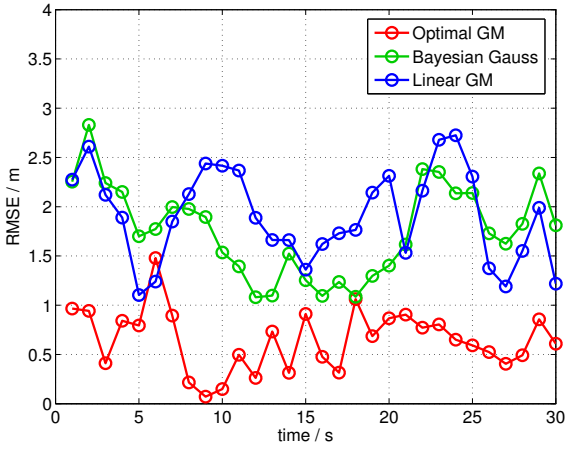

Figure 2. A comparison of the state estimation RMSE as a function of time.

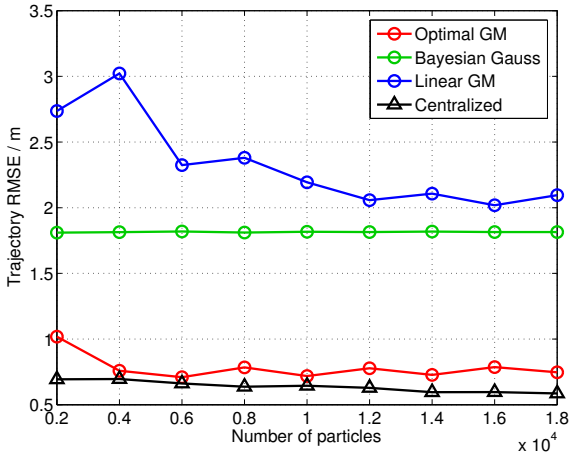

Figure 3. A comparison of the trajectory estimation RMSE as a function of the number of particles

S_k was located at $\boldsymbol{l}_k = (l_{k,1}, l_{k,2})$ with the observation function

$$\boldsymbol{h}_k(\boldsymbol{x}_n) = \begin{bmatrix} \sqrt{(x_{n,1}-l_{k,1})^2 + (x_{n,2}-l_{k,2})^2} \\ \arctan\left[(x_{n,2}-l_{k,2})/(x_{n,1}-l_{k,1})\right] \end{bmatrix} \quad (31)$$

and the observation noise

$$\boldsymbol{v}_{n,k} \sim \begin{bmatrix} 0.5\mathcal{N}(\mu_{v,1}, \sigma_{v,1}^2) + 0.5\mathcal{N}(-\mu_{v,1}, \sigma_{v,1}^2) \\ 0.5\mathcal{N}(\mu_{v,2}, \sigma_{v,2}^2) + 0.5\mathcal{N}(-\mu_{v,2}, \sigma_{v,2}^2) \end{bmatrix}. \quad (32)$$

In the examples, we assumed $\sigma_u = 0.71$, $\mu_{v,1} = \mu_{v,2} = 0.7$, and $\sigma_{v,1} = \sigma_{v,2} = 0.55$. Also, we assumed $\boldsymbol{x}_0 \sim \mathcal{N}\left([0\text{m}, 0\text{m}, 1\text{m/s}, 1\text{m/s}]^T, \boldsymbol{R}\right)$ as the prior information. Moreover, we used $\epsilon = 1/6$ for average consensus, 4 components for Gaussian mixtures, 8000 samples for fusion of Gaussian mixtures, and 10000 particles for filtering, unless otherwise stated.

In Fig. 2 we show the state estimation root-mean-square errors (RMSE) of three posterior-based distributed particle filtering algorithms as a function of time. According to the figure, at almost every time point our method ("optimal GM") yielded a significantly lower RMSE than Bayesian fusion of Gaussian approximations ("Bayesian Gauss") and linear fusion of Gaussian mixtures ("linear GM") did.

Fig. 3 compares the trajectory estimation RMSEs of the three methods with that of centralized particle filtering, using different numbers of particles. As shown in the figure, the RMSEs of our method, more than 50% lower than those of the other two methods, were very close to those of the centralized method. Also, we notice that all the methods, except for the Bayesian Gauss method, tended to have higher estimation accuracy with a larger number of particles used, which coincides with the common sense that more particles generally bring more accurate approximations. The Bayesian Gauss method was almost insensitive to the number of particles, because all it needs from an ensemble of particles is its mean and variance (covariance matrix), neither of which varies much with the size of the ensemble, as long as the size is not too small. Our method also showed a certain degree of robustness to the number of particles.

Fig. 4 shows the process of consensus within a randomly picked particle filtering iteration. As shown in the figure, the average Kullback–Leibler (KL) distance between estimates of neighboring sensors dropped, and the average RMSE of local estimates decreased, across iterations of the consensus algorithm. Both of them converged while fluctuating because of the randomness involved in the Monte Carlo method. As we can see, the final average KL distance was very close to zero, i.e., exact consensus, but not able to reach it because of the errors discussed in Section V-B. The final average RMSE was also very close to that of the centralized method, which, again, verifies the effectiveness of the proposed method to approximate the centralized approach in a distributed fashion.

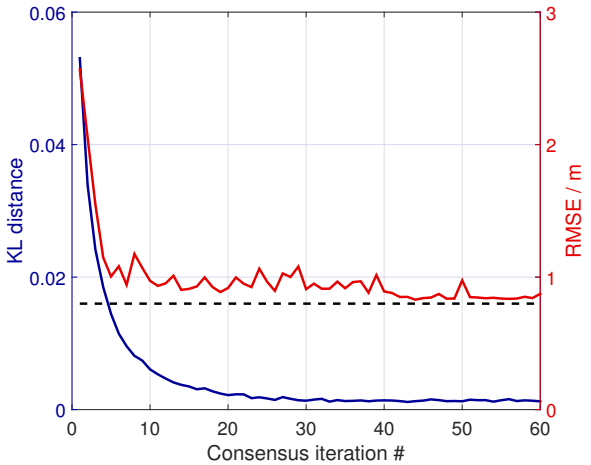

Figure 4. Average KL distance and RMSE across iterations of consensus

VII. Conclusion

In this paper, we proposed a distributed particle filtering algorithm based on optimal fusion of local posterior estimates approximated as Gaussian mixtures. Since a local posterior estimate exists as an ensemble of weighted particles in particle filtering, we proposed to learn a Gaussian mixture directly from weighted samples. We implemented the optimal fusion rule in a distributed fashion, using an average consensus algorithm. We derived a distributed fusion rule for the consensus algorithm, and performed the fusion of Gaussian mixtures via importance sampling. As we can see, the posterior-based consensus process involves no observation functions, thus allowing diverse sensing modalities in the network. With the extra variables and constants involved in consensus, convergence of the proposed distributed particle filtering algorithm does not directly follow from that of an average consensus algorithm in its canonical form. We therefore proved the convergence of the proposed algorithm, which was then validated by numerical examples. It was also demonstrated that the proposed algorithm yields significantly higher estimation accuracy than other posterior-based distributed particle filtering algorithms.

The Gaussian mixture model used in the proposed algorithm has a compact parametric representation, which is intended for low communication overhead. The overhead is, however, roughly proportional to the number of components in a Gaussian mixture. Hence, the optimal number of components is one that balances approximation accuracy and communication overhead, and often varies for different sensors. The proposed algorithm gives each sensor the flexibility of choosing its own optimal number of components, because the fusion of Gaussian mixtures does not put any constraints on their numbers of components. In future work, we will explore efficient methods to learn a Gaussian mixture with an optimal number of components from weighted samples. Also, we will study analytical approaches to fusion of Gaussian mixtures.

Acknowledgement

This work was supported by the AFOSR Grant No. FA9550-11-1-0210.

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, Feb. 2002.

[2] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[3] P. M. Djurić, M. F. Bugallo, and J. Miguez, "Density assisted particle filters for state and parameter estimation," in *ICASSP 2004*, Montreal, Quebec, Canada, May 2004, pp. II – 701 – 704.

[4] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215 – 233, Jan. 2007.

[5] D. Üstebay, M. Coates, and M. Rabbat, "Distributed auxiliary particle filters using selective gossip," in *Proc. IEEE ICASSP*, Prague, Czech Republic, May 2011, pp. 3296 – 3299.

[6] C. J.Bordin and M. G. S. Bruno, "Consensus-based distributed particle filtering algorithms for cooperative blind equalization in receiver networks," in *Proc. IEEE ICASSP*, Prague, Czech Republic, May 2011, pp. 3968 – 3971.

[7] Y.-C. Wu, Q. Chauhari, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 28, pp. 124 – 138, Jan. 2011.

[8] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz, "Distributed synchronization in wireless networks," *IEEE Signal Processing Magazine*, vol. 25, pp. 81 – 97, Sep. 2008.

[9] J. Li and A. Nehorai, "Joint sequential target state estimation and clock synchronization in wireless sensor networks," in *Proc. 48th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2014.

[10] O. Hlinka, F. Hlawatsch, and P. M. Djurić, "Likelihood consensus-based distributed particle filtering with distributed proposal density adaptation," in *Proc. IEEE ICASSP*, Kyoto, Japan, Mar. 2012, pp. 3869–3872.

[11] O. Hlinka, O. Slučiak, F. Hlawatsch, P. M. Djurić, and M. Rupp, "Likelihood consensus and its application to distributed particle filtering," *IEEE Trans. Signal Processing*, vol. 60, no. 8, pp. 4334 – 4349, Aug. 2012.

[12] O. Slučiak, O. Hlinka, M. Rupp, F. Hlawatsch, and P. M. Djurić, "Sequential likelihood consensus and its application to distributed particle filtering with reduced communications and latency," in *Proc. 45th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2011, p. 1766–1770.

[13] B. N. Oreshkin and M. J. Coates, "Asynchronous distributed particle filter via decentralized evaluation of Gaussian products," in *Proc. Information Fusion*, Edinburgh, U.K., July 2010, pp. 1–8.

[14] A. Mohammadi and A. Asif, "Consensus-based distributed unscented particle filter," in *Proc. IEEE Statistical Signal Processing Workshop*, Nice, France, June 2011, pp. 237–240.

[15] D. Gu, J. Sun, Z. Hu, and H. Li, "Consensus based distributed particle filter in sensor networks," in *Proc. IEEE Int'l Conference on Information and Automation*, Changsha, China, June 2008, pp. 302 – 307.

[16] D. Gu, "Distributed particle filter for target tracking," in *Proc. IEEE Int'l Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, p. 3856–3861.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.

[18] J. V. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Wiley-Interscience, 2009.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 39, no. 1, pp. 1 – 38, 1977.