Spring 5-15-2016

Distributed Target Tracking and Synchronization in Wireless Sensor Networks

Jichuan Li
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Electrical & Systems Engineering

Dissertation Examination Committee:
Arye Nehorai, Chair
R. Martin Arthur
Nan Lin
Hiro Mukai
Heinz Schaettler

Distributed Target Tracking and Synchronization in Wireless Sensor Networks

by

Jichuan Li

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

May 2016
Saint Louis, Missouri

Contents

List of Figures

List of Tables

Acknowledgments

I am sincerely grateful to my research advisor, Dr. Arye Nehorai, for his continuous guidance and support for my research at Washington University and my career after graduation. He has given me a lot of freedom to explore research topics that interest me the most, and supported my exploration with advice and encouragement.

I wish to thank my dissertation defense committee members, Dr. R. Martin Arthur, Dr. Nan Lin, Dr. Hiro Mukai, and Dr. Heinz Schaettler, and my preliminary research exam committee members, Dr. R. Martin Arthur and Dr. Hiro Mukai, for their valuable suggestions.

I am also thankful to my labmates, Phani, Elad, Peng, Xiaoxiao, Keyong, Zhao, Alex, Mengxue, Mianzhi, Prateek, Zhen, Yijian, and Hesam, for their help and support.

I owe my deepest thanks to my beloved parents for their unselfish love.

Jichuan Li

Washington University in Saint Louis

May 2016

Dedicated to my parents

ABSTRACT OF THE DISSERTATION

Distributed Target Tracking and Synchronization in Wireless Sensor Networks

by

Jichuan Li

Doctor of Philosophy in Electrical Engineering

Washington University in St. Louis, 2016

Professor Arye Nehorai, Chair

Wireless sensor networks provide useful information for various applications but pose challenges in scalable information processing and network maintenance. This dissertation focuses on statistical methods for distributed information fusion and sensor synchronization for target tracking in wireless sensor networks.

We perform target tracking using particle filtering. For scalability, we extend centralized particle filtering to distributed particle filtering via distributed fusion of local estimates provided by individual sensors. We derive a distributed fusion rule from Bayes' theorem and implement it via average consensus. We approximate each local estimate as a Gaussian mixture and develop a sampling-based approach to the nonlinear fusion of Gaussian mixtures. By using the sampling-based approach in the fusion of Gaussian mixtures, we do not require each Gaussian mixture to have a uniform number of mixture components, and thus give each sensor the flexibility to learn a Gaussian mixture model consisting of an optimal number of mixture components, based on its local information. Given such flexibility, we

develop an adaptive method for Gaussian mixture learning through a combination of hierarchical clustering and the expectation-maximization algorithm. Using numerical examples, we show that the proposed distributed particle filtering algorithm improves the accuracy and communication efficiency of distributed target tracking, and that the proposed adaptive Gaussian mixture learning method improves the accuracy and computational efficiency of distributed target tracking.

We also consider the synchronization problem of a wireless sensor network. When sensors in a network are not synchronized, we model their relative clock offsets as unknown parameters in a state-space model that connects sensor observations to target state transition. We formulate the synchronization problem as a joint state and parameter estimation problem and solve it via the expectation-maximization algorithm to find the maximum likelihood solution for the unknown parameters, without knowledge of the target states. We also study the performance of the expectation-maximization algorithm under the Monte Carlo approximations used by particle filtering in target tracking. Numerical examples show that the proposed synchronization method converges to the ground truth, and that sensor synchronization significantly improves the accuracy of target tracking.

Chapter 1

Introduction

In recent years, wireless sensor networks have emerged as a widely used tool in various applications. One of the major applications is target tracking, which benefits from the diverse perspectives of observations provided by a network of different sensors. In order to collect more information, a large-scale sensor network can be used. Although more information means more confidence in estimation, a large-scale network demands scalable signal processing. For this reason, distributed target tracking has become an important research area to explore. Also, for a large network, it is difficult to keep all the sensors perfectly synchronized, which is extremely important for time-sensitive applications like target tracking. Thus, it is also necessary to keep track of the relative clock offsets between sensors from time to time. In this dissertation, we focus on two topics related to target tracking using wireless sensor networks. In the first topic, we study distributed particle filtering for distributed target tracking; in the second topic, we develop statistical methods for sensor synchronization.

1.1 Background

Recursive Bayesian estimation is a powerful statistical approach to target tracking. Generally, a recursive Bayesian estimation algorithm sequentially updates its belief in the current target state based on its previous belief and the incoming information. The Kalman filter is one of the most popular recursive Bayesian estimation algorithms and has been successfully used in various applications. However, a Kalman filter relies on two key assumptions of the dynamic model. First, it assumes both the state transition model and the observation model to be linear; second, it assumes both the state transition noise and the observation noise to be Gaussian. In practice, however, these two assumptions might not be always satisfied, and in this case, a Kalman filter would not work. To avoid the limitation of the Kalman filter, the extended Kalman filter (EKF) was developed. An extended Kalman filter still assumes Gaussian noise but linearizes the dynamic model through the Taylor series expansion, which unfortunately works effectively only under slight nonlinearity. To push the limits forward, the unscented Kalman filter (UKF) was developed based on the so-called unscented transform. Thanks to the unscented transform, an unscented Kalman filter works well with nonlinear models but still relies on the assumption of Gaussian noise. To deal with both nonlinear models and non-Gaussian noise, a particle filter (PF) was developed, based on Monte Carlo approximations. Thanks to the Monte Carlo approximations, the particle filter is flexible enough to work effectively with both nonlinear models and non-Gaussian noise, and replaces the Kalman filter family when their assumptions are not completely satisfied.

To guarantee its estimation accuracy under the existence of noise in a dynamic model, recursive Bayesian estimation often needs multiple observations, possibly from different perspectives, at each time step. In practice, we often obtain observations from multiple sensors and use a wireless sensor network to collect multiple observations at each time step. A wireless

sensor network can be implemented in a centralized way with a fusion center that receives observations from every sensor in the network and processes them in batches. A centralized implementation works conveniently when the network is small, but shows limitations when the network is large. First, it has a single point of failure. If the fusion center fails, the whole network fails. Second, it makes communication difficult. A distant sensor needs other sensors to relay its message to the fusion center, which necessitates routing. When some sensor fails or the topology of the network changes, the previous routing strategy might be no longer valid, and a new one has to be designed, thus adding difficulty in maintenance. Third, as a side effect of centralized routing, sensors located near the fusion center relay considerably more messages and thus consume significantly more energy than those located far away. Such unbalanced energy consumption affects the longevity of a wireless sensor network. All these limitations prevent a centralized implementation from scaling. To avoid these limitations, we pursue a distributed implementation, in which every sensor processes its own observation and repeatedly communicates with its neighbors to fuse their local results together. A distributed implementation has information stored in every sensor in the network and thus is robust to failures. Also, it requires communications between neighbors only and thus does not need routing, which leads to balanced energy consumption across the network and robustness to changes in the network topology.

1.1.1 Distributed particle filtering

A particle filter implemented in a distributed fashion is called distributed particle filtering (DPF). In this dissertation, we study distributed particle filtering based on the fusion of local posterior density functions, which is also known as posterior-based distributed particle filtering. In posterior-based distributed particle filtering, each sensor performs local particle

filtering based on its own observation and obtains a local posterior during each time step. Then, each sensor iteratively communicates its posterior with its neighbors and updates its posterior based on those received from its neighbors, until convergence. There are two problems to solve in posterior-based distributed particle filtering, namely how to parametrically represent each posterior for wireless transmission and how to fuse the parametrically represented posteriors in a distributed fashion. In the literature, there seems to be a trade-off in solving these two problems. For accurate parametric approximations, an accurate fusion rule becomes intractable, and thus a suboptimal rule has to be used; similarly, for an accurate fusion rule, an over-simplified parametric approximation approach has to be taken. As a result, posterior-based distributed particle filtering is often not sufficiently accurate, although it has many other advantages, which are introduced in Section 2.

1.1.2 Sensor synchronization

Wireless sensor networks are powerful because of the collaboration among different sensors. For time-sensitive tasks like target tracking, however, effective collaboration strongly relies on perfect synchronization between sensors, which is usually difficult to maintain. The internal clock of a sensor can drift away from its initial setting, due to random mechanical issues or environmental impacts. Due to imperfect synchronization, sensors programmed to observe a target at a given time might end up with observations taken at different times. If we simply ignore the fact that a sensor network is unsynchronized, our inference from the asynchronous sensor observations would be inaccurate. To solve this problem, we need to learn the relative clock offset between sensors. A mainstream approach to sensor synchronization is to ask a sensor about its time. The time of a sensor is usually communicated to other sensors via messages that convey timestamps. If we do not have any other useful information, it is

reasonable to create and communicate timestamps for sensor synchronization; otherwise, it is preferable to infer from information we have, so that we do not have to waste the energy of each sensor in creating and communicating extra information. As we know, when a wireless sensor network tracks a moving target, sensor observations are a function of time and are also correlated due to the common target. For this reason, sensor observations implicitly tell us when they were actually taken, and by combining observations from different sensors together, we can learn their relative clock offsets without collecting additional information. In this dissertation, we study statistical signal processing methods for sensor synchronization in target tracking.

1.2 Contributions of this work

This dissertation studies posterior-based distributed particle filtering and statistical sensor synchronization. We summarize the main contributions as follows.

Optimal distributed fusion: We develop a distributed fusion rule for local posteriors, which is optimal from a Bayesian perspective. The distributed fusion rule is obtained through average consensus from a centralized fusion rule, which stems from Bayes' theorem. Because a posterior density function is always normalized, the average consensus algorithm involves an additional normalization step in each consensus iteration. We prove that the average consensus algorithm with the additional normalization step still converges to the desired global average.

Adaptive Gaussian mixture learning: We represent each posterior as a Gaussian mixture model, a convex combination of multiple Gaussian components. A Gaussian mixture

model can be learned via the expectation-maximization (EM) algorithm from (weighted) samples representing a posterior. However, the EM algorithm does not make an adaptive decision about the number of components in a Gaussian mixture model. Also, the EM algorithm is computationally intensive and sometimes needs a large number of iterations to converge. To enable adaptivity and guarantee computational efficiency, we develop a divisive hierarchical clustering algorithm based on an adaptive version of the principal component partitioning (PCP) tree. We apply this algorithm to learn from weight samples a Gaussian mixture model consisting of an adaptively determined number of components. We also send the obtained Gaussian mixture model as an initial guess to the EM algorithm, and show that the EM algorithm needs only a small number of iterations to move the current model to a maximum likelihood solution.

Importance sampling for nonlinear fusion: The optimal fusion rule is nonlinear and thus makes the fusion of Gaussian mixtures analytically intractable. We propose an importance sampling approach to the nonlinear fusion problem. Instead of using a single proposal distribution, we consider each Gaussian mixture to be fused as one of our proposal distributions, and draw samples from each Gaussian mixture. To take into consideration the contribution of each Gaussian mixture to the fusion, we allocate to each Gaussian mixture a number of samples proportional to its contribution.

Clock synchronization for target tracking: We develop a hypothesis testing approach to learn the temporal order of sensor clocks. Based on the temporal order, we build a unsynchronized multi-sensor state-space model to connect asynchronous sensor observations with the underlying target states. Under the built model, we solve the joint estimation problem via a stochastic variant of the EM algorithm and analyze the performance of the solution under Monte Carlo approximations.

1.3 Organization of the dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we develop a posterior-based distributed particle filtering framework via average consensus. In Chapter 3, we design an adaptive Gaussian mixture learning algorithm to be used in the distributed particle filtering framework developed in Chapter 2. In Chapter 4, we solve the sensor synchronization problem in target tracking. In Chapter 5, we summarize our contributions and discuss future directions.

Chapter 2

Distributed Particle Filtering via Average Consensus

In this chapter, we propose a distributed particle filtering algorithm based on optimal fusion of local posteriors. We derive an optimal fusion rule from Bayes' theorem, and implement it in a distributed and iterative fashion via an average consensus algorithm. We approximate local posteriors 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 locally available at each sensor in the network. Numerical examples demonstrate the advantages of the proposed method in estimation accuracy and communication efficiency over other distributed particle filtering algorithms.[1]

[1]This chapter is based on J. Li and A. Nehorai, "Distributed particle filtering via optimal fusion of Gaussian mixtures," in *18th International Conference on Information Fusion*, Washington D.C., July 2015, pp. 1182–1189.

2.1 Introduction

Distributed particle filtering consists of separate particle filters that have access to local observations only and produce global estimates via distributed fusion. It is often implemented using a consensus algorithm [1], where sensors in a network reach agreement among their beliefs iteratively through communications between neighboring sensors. Depending on the type of information communicated between sensors in average consensus, a distributed particle filtering algorithm can be categorized as particle-based, likelihood-based, or posterior-based.

Particle-based algorithms [2]–[6] communicate the local likelihood or the local weight of each particle. To guarantee an accurate Monte Carlo approximation, the number of particles needed in each local filter is usually quite large, which results in considerably high communication overhead, if the information of every single particle is transmitted. Also, in order to fuse particle information, each local filter must have an identical set of randomly-generated particles, which necessitates perfect synchronization between the random number generators in different sensors. The reliance on perfect synchronization, together with the high communication overhead, makes particle-based algorithms costly to implement in practice.

Likelihood-based algorithms [7]–[9] communicate local likelihood functions parametrically approximated via factorization and linear regression. Since there is no universal approach to the desired factorization, the likelihood approximation approach does not generalize well beyond the exponential family. Also, likelihood consensus requires uniform factorization across the network and thus does not apply to scenarios where the noise distribution at each sensor varies. Hence, likelihood consensus might not be an ideal choice for general applications.

Posterior-based algorithms [10]–[13] communicate local posteriors parametrically approximated in a compact form, and have several advantages over likelihood-based and particle-based algorithms. First, unlike likelihood functions, posteriors are essentially density functions and thus easy to represent parametrically. If a posterior follows a (multivariate) Gaussian distribution, it can be losslessly represented by its mean and variance (covariance matrix); if a posterior 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) [14]. Also, such a compact parametric representation incurs significantly lower communication overhead than a nonparametric representation, e.g., particles. Moreover, posterior-based algorithms are invariant to how local posteriors are obtained and thus allow diverse sensing modalities [15] and various filtering tools to be exploited in a network. Last but not least, posterior-based algorithms give each sensor privacy, since no sensor in the network needs to know how any other obtains its local posterior.

The challenge of posterior-based algorithms mainly lies in the fusion of parametrically-represented local posteriors. In [10] and [11], local posteriors are fused in a Bayesian fashion but assumed to be Gaussian for fusion tractability. As we know, a posterior follows a Gaussian distribution only if both the state transition model and the observation model are linear with additive Gaussian noise. Thus, the Gaussian assumption is so strong that it will incur obvious approximation errors in applications with nonlinear models or non-Gaussian noise. In [12] and [13], local posteriors are approximated as Gaussian mixtures but fused linearly through their parameters. This linear fusion rule is, however, suboptimal because it is not justified by the underlying statistical model. Also, it requires Gaussian mixtures to have a uniform number of components, thus limiting the flexibility and adaptivity of local parametric representation. In [11] and [12], unscented particle filters [16], a category of particle filters with a special proposal distribution, are used for local filtering. However, since fusion

and local filtering in posterior-based algorithms are separated and do not interfere with each other, unscented particle filters do not actually contribute to the fusion process.

In this chapter, we propose a posterior-based distributed particle filtering algorithm. We approximate local posteriors as Gaussian mixtures and fuse local posteriors via an optimal distributed fusion rule derived from Bayes' theorem and implemented via an average consensus algorithm. Unlike other posterior-based algorithms, the proposed algorithm neither compromises approximation accuracy for fusion tractability nor compromises fusion validity for approximation accuracy. Also, the proposed algorithm seeks consensus on the posterior distribution represented by a parametric approximation, rather than on the parametric approximation itself, thus giving flexibility to local parametric approximations by allowing each Gaussian mixture to have an optimal yet possibly nonuniform number of components. To address the challenge in fusion, we design algorithms based on importance sampling to fuse Gaussian mixtures nonlinearly within each consensus step. Finally, we prove the convergence of the proposed distributed particle filtering algorithm and demonstrate its advantages in both estimation accuracy and communication efficiency through numerical examples.

Note that consensus is not the only way to implement distributed particle filtering. For example, in [17]–[20], a leading sensor or a chain of leading sensors is selected during each time step to perform filtering; in [21], a diffusion-based scheme is proposed to reduce the communication overhead in a consensus-based algorithm. However, in this chapter, we focus on consensus-based distributed particle filtering only, given the advantages elaborated in Section 2.4.

The rest of the chapter is organized as follows. Section 2.2 introduces a network model and a state-space model. Section 2.3 introduces centralized particle filtering. Section 2.4 presents our distributed particle filtering algorithm. Section 2.5 analyzes the performance of

the proposed algorithm. Section 2.6 presents numerical examples, and Section 2.7 concludes the chapter.

2.2 Problem formulation

2.2.1 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, corresponding to sensors, with cardinality $|\boldsymbol{V}| = K$, and $\boldsymbol{E} \subset \boldsymbol{V} \times \boldsymbol{V}$ is the set of edges, corresponding to communication links between sensors. We assume each communication link to be bidirectional, in the sense that sensors can transmit information in either direction through the link. With no particular direction assigned to any edge, we assume the graph $\boldsymbol{G}$ to be undirected. We consider two different sensors as neighbors if and only if their distance is below a threshold ρ, and assume the existence of a communication link between two sensors if and only if they are neighbors. We define the neighborhood of a sensor as the set of all its neighbors plus the sensor itself. We assume that the graph $\boldsymbol{G}$ is connected, or in other words that there exists a multi-hop communication route connecting any two sensors in the network. Moreover, we assume the sensor network to be synchronized; otherwise, we synchronize the network via a clock synchronization scheme [22]–[24].

2.2.2 Signal model

We consider a single moving target to be observed by the sensor network. We connect target state transition with sensor observation using a discrete-time state-space model,

$$\begin{cases} \boldsymbol{x}_n = \boldsymbol{g}(\boldsymbol{x}_{n-1}) + \boldsymbol{u}_n \\ \boldsymbol{y}_{n,k} = \boldsymbol{h}_k(\boldsymbol{x}_n) + \boldsymbol{v}_{n,k} \quad (k = 1, 2, \ldots, K) \end{cases}, \tag{2.1}$$

where

1) $\boldsymbol{x}_n \in \mathbb{R}^d$ is the target state at the nth time point;

2) $\boldsymbol{y}_{n,k} \in \mathbb{R}^{b_k}$ is the observation taken by S_k at the nth time point;

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 noises;

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.

2.2.3 Goal

The goal is to sequentially estimate the current state $\boldsymbol{x}_n$ based on the estimate of the preceding state $\boldsymbol{x}_{n-1}$ and the newly available observations $\{\boldsymbol{y}_{n,1}, \boldsymbol{y}_{n,2}, \ldots, \boldsymbol{y}_{n,K}\}$.

2.2.4 Notations

We denote consecutive states $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ as $\boldsymbol{x}_{1:n}$, observations taken by the whole network at the nth time point $\{\boldsymbol{y}_{n,1}, \boldsymbol{y}_{n,2}, \ldots, \boldsymbol{y}_{n,K}\}$ as $\boldsymbol{y}_n$, and consecutive observations taken by the whole network $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ 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 in importance sampling. We denote the neighborhood of S_k as N_k.

2.3 Centralized particle filtering

The problem formulated in Section 2.2 is a filtering problem. A filtering problem is often solved by a particle filter when the state-space model is nonlinear or the noise is 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):

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \approx \sum_{m=1}^{M} w_n^{(m)} \delta(\boldsymbol{x}_n - \boldsymbol{x}_n^{(m)}), \tag{2.2}$$

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 [25], a particle is generated according to a proposal distribution $q(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}^{(m)}, \boldsymbol{y}_n)$, and its weight is

updated according to

$$w_n^{(m)} \propto \frac{f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)})f(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{n-1}^{(m)})}{q(\boldsymbol{x}_n^{(m)}|\boldsymbol{x}_{n-1}^{(m)}, \boldsymbol{y}_n)} \times w_{n-1}^{(m)}. \tag{2.3}$$

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

$$w_n^{(m)} \propto f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)}) \times w_{n-1}^{(m)}. \tag{2.4}$$

The global likelihood function $f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)})$ in (2.4) can be factorized into a product of local likelihood functions,

$$f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)}) = \prod_{k=1}^{K} f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n^{(m)}), \tag{2.5}$$

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 falls below a threshold, a possible remedy is to resample the particles according to their weights. A popularly used estimate of the effective sample size of an ensemble is

$$\hat{M}_e = \left[\sum_{m=1}^{M} (w_n^{(m)})^2\right]^{-1}, \tag{2.6}$$

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 in utilizing sensor observations, 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. Second, it causes severely unbalanced energy consumption and communication traffic in the network, because sensors located near the fusion center relay many more messages than those located far away. Further, reliance on a single fusion center makes it vulnerable to a single point of failure. Moreover, it does not scale with the network size. Therefore, it is often preferable to perform distributed particle filtering.

2.4 Distributed particle filtering

In distributed particle filtering, every sensor in the network performs local particle filtering on its own observation while communicating with its neighbors for information fusion, thus achieving centralized filtering in a distributed fashion.

2.4.1 Consensus

Consensus [1] is a type of information fusion algorithm in which every sensor in the network iteratively communicates with its neighbors and updates its own belief based on its neighbors' until all the sensors hold the same belief. Consensus has the following advantages in distributed data fusion. First, it ends up with a global estimate available at each sensor in the network, so that the network is robust to sensor failures and every sensor in the network is ready to react based on the global estimate. Second, it requires only local communications and does not need global routing. Last but not least, it is robust to changes in the network

topology. In this chapter, we fuse local posteriors provided by different sensors via consensus, so that every sensor in the network ultimately obtains a global posterior.

Likelihood factorization in (2.5), as mentioned in Section 2.3, makes data fusion convenient, because its logarithmic form

$$\log f(\boldsymbol{y}_n|\boldsymbol{x}_n) = \sum_{k=1}^{K} \log f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) \tag{2.7}$$

gives rise to a straightforward implementation of an average consensus algorithm [1]. However, unlike a prior or posterior density function, a likelihood function is generally difficult to approximate parametrically through a universal approach such as the Gaussian mixture model. This difficulty motivates us to communicate posterior density functions, instead of likelihood functions, in an average consensus algorithm.

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

$$f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) = f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n, \boldsymbol{y}_{1:n-1}), \tag{2.8}$$

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

$$f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) = \frac{f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1}) f(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})}. \tag{2.9}$$

Substitute (2.9) into (2.7), and then we get

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

which simplifies to

$$\log f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) + (K-1)\log f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \sum_{k=1}^{K} \log f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1}) + \text{const.} \tag{2.10}$$

The constant term in (2.10) 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.

Equation (2.10) presents a centralized fusion rule for local posteriors: $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$ on the right-hand side of (2.10) is the local posterior of $\boldsymbol{x}_n$ by S_k, while $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$ on the left-hand side of (2.10) is the global posterior of $\boldsymbol{x}_n$ by the whole network. There are two other terms in (2.10), namely the constant term and the prediction term. The constant term will disappear when we normalize $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$ so that it integrates to 1; the prediction term $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$ can be calculated as

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \int f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1}) \mathrm{d}\boldsymbol{x}_{n-1}, \tag{2.11}$$

where $f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})$ is available from the state transition model, and $f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})$, i.e., the global posterior of the last state, is available at each sensor thanks to the consensus algorithm performed during the last time step.

The centralized fusion rule (2.10) can be implemented in a distributed fashion through an average consensus algorithm. Denote $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$ as $\eta_k^{(0)}(\boldsymbol{x}_n)$, and then the summation on the right-hand side of (2.10) can be computed iteratively based on a two-step distributed

fusion rule:

$$\text{Step 1: } \log \eta_k^{(i+1)}(\boldsymbol{x}_n) = \sum_{j \in N_k} \varepsilon_{kj} \log \eta_j^{(i)}(\boldsymbol{x}_n), \tag{2.12}$$

$$\text{Step 2: Normalize } \eta_k^{(i+1)}(\boldsymbol{x}_n), \tag{2.13}$$

where $\eta_k^{(i)}(\boldsymbol{x}_n)$ is the posterior density function of $\boldsymbol{x}_n$ held by S_k in the ith iteration of the average consensus algorithm during the nth time step, N_k is the neighborhood of S_k with S_k included, and ε_{kj} is the Metropolis weight [26] defined as

$$\varepsilon_{kj} = \begin{cases} 1/\max\{|N_k|, |N_j|\} & \text{if } (k,j) \in \boldsymbol{E} \\ 1 - \sum_{l \in N_k} \varepsilon_{kl} & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}. \tag{2.14}$$

We call (2.12) the distributed fusion step and (2.13) the normalization step. In the distributed fusion step, every sensor iteratively sends its current belief to its neighbors and updates it based on beliefs received from its neighbors; in the normalization step, a updated belief is normalized so that it appears as a valid probability density function and can be compactly approximated by a Gaussian mixture model for communication.

In Section 2.5.1, we show that under certain conditions, we have for $\forall\ k$

$$\lim_{i \to \infty} \log \eta_k^{(i)}(\boldsymbol{x}_n) = \frac{1}{K} \sum_{j=1}^{K} \log \eta_j^{(0)}(\boldsymbol{x}_n) + \text{const.} \tag{2.15}$$

Combining (2.10) and (2.15), we have for $\forall\ k$

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \propto \frac{\left(\lim_{i\to\infty} \eta_k^{(i)}(\boldsymbol{x}_n)\right)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}}, \tag{2.16}$$

which, called the recovery step, concludes the consensus-based distributed particle filtering.

2.4.2 Gaussian mixture model

Consensus necessitates inter-sensor communication. Communication is a major source of energy consumption for 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 this chapter, we compress all the posteriors in the distributed fusion step (2.12) and the recovery step (2.16) into Gaussian mixtures [14].

A Gaussian mixture is a convex combination of Gaussian components as follows,

$$\eta_k^{(i)}(\boldsymbol{x}_n) \approx \sum_{c=1}^{C} \alpha_c \, \mathcal{N}\left(\boldsymbol{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right), \tag{2.17}$$

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

A Gaussian mixture model can be used to approximate an arbitrary probability distribution, and is often learned via the expectation-maximization (EM) algorithm [27] from samples representing the underlying distribution. In particle filtering, samples are often weighted

Algorithm 1: GM Learning from Weighted Samples

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

due to importance sampling, and thus we learn a Gaussian mixture model directly from weighted samples using the weighted EM algorithm [28], as summarized in Algorithm 1.

The convergence of Algorithm 1 can be determined in different ways. In this chapter, we terminate Algorithm 1 when the absolute difference between the log-likelihoods of the current and previous Gaussian mixture models is smaller than a chosen percentage of that between the log-likelihoods of the current and initial models.

In Algorithm 1, the number of components C has to be specified during initialization before learning. Generally, a mixture of more components tends to provide a more accurate approximation but lead to higher communication overhead. The optimal number of components should strike a balance between approximation accuracy and communication cost.

Methods to determine the optimal number of components abound [29]–[32] but are beyond the scope of this chapter. In this chapter, we use a predefined number of components C, randomly divide the samples into C groups of (approximately) equal size, and initialize the EM algorithm with the statistics of the C groups.

Note that the number of components can even be one, which results in a Gaussian distribution as a special case. In this sense, if the number of components is adaptively determined, the approximation accuracy of a Gaussian mixture would not be lower than that of a Gaussian distribution.

2.4.3 Fusion of Gaussian mixtures

With posteriors approximated as Gaussian mixtures, the fusion of Gaussian mixtures has to be considered for both the distributed fusion step (2.12) and the recovery step (2.16). For convenience, we convert the distributed fusion step (2.12) from the logarithmic form to the exponential form:

$$\eta_k^{(i+1)}(\boldsymbol{x}_n) = \prod_{j \in N_k} \left(\eta_j^{(i)}(\boldsymbol{x}_n) \right)^{\varepsilon_{kj}}. \tag{2.18}$$

Both (2.16) and (2.18) involve a product of powers of Gaussian mixtures, which is unfortunately intractable to compute analytically. Therefore, we consider importance sampling. In [33], various methods are proposed to sample from a product of Gaussian mixtures, but unfortunately none of them directly applies to our problem, because of the negative exponents in (2.16) and the fractional exponents in (2.18). In this chapter, we extend the mixture importance sampling approach presented in [33] to a general case where Gaussian

mixtures in the product have fractional or negative exponents, and propose a weighted mixture importance sampling approach to the fusion of Gaussian mixtures in both (2.16) and (2.18).

Distributed fusion step

We generate samples from each Gaussian mixture to be fused and assign to them importance weights calculated under their corresponding proposal distributions. For each $j \in N_k$, we draw M_j samples $\{\boldsymbol{x}_n^{(j,m)}\}_{m=1}^{M_j}$ from $\eta_j^{(i)}(\boldsymbol{x}_n)$ and assign to each $\boldsymbol{x}_n^{(j,m)}$ an importance weight $w_n^{(j,m)}$ calculated as

$$w_n^{(j,m)} = \left(\eta_j^{(i)}(\boldsymbol{x}_n^{(j,m)})\right)^{-1} \prod_{l \in N_k} \left(\eta_l^{(i)}(\boldsymbol{x}_n^{(j,m)})\right)^{\varepsilon_{kl}}. \tag{2.19}$$

We set M_j to be proportional to the Metropolis weight ε_{kj}, i.e., $M_j = \lfloor M\varepsilon_{kj} \rfloor$, where $\lfloor\ \rfloor$ is the floor function and M is the given total number of samples to be drawn (the total number of thus generated samples might be smaller than M due to rounding, but could be manually adjusted back to M by distributing the unused quota to some of the Gaussian mixtures to be fused). After applying the normalization step (2.13) to $\{w_n^{(j,m)}\}$, a Gaussian mixture model of the updated posterior $\eta_k^{(i+1)}(\boldsymbol{x}_n)$ can be learned from the weighted samples $\{\boldsymbol{x}_n^{(j,m)}, w_n^{(j,m)}\}$ using Algorithm 1.

Here, the proposed approach draws samples from each Gaussian mixture to be fused, so that the drawn samples cover most of the support of the fused density function. Since multiple proposal distributions are used, the proposal approach equivalently samples from a mixture of proposal distributions. However, we do not have to use the whole mixture when calculating the importance weight of a sample. Instead, since we know exactly which

proposal distribution in the mixture each sample is drawn from, it would be more accurate if we use the corresponding proposal distribution alone when calculating the importance weight. Also, since the sampling bias introduced by each proposal distribution is eliminated when we divide the true density by the corresponding importance density, importance weights calculated under different proposal distributions are consistent.

Another contribution of the proposed approach is weighted sample allocation. As we can see, the Gaussian mixtures in (2.18) do not contribute equally to the product, and a Gaussian mixture with a large exponent contributes more to the product and is more influential in local fusion than one with a small exponent. By adjusting the contribution of each Gaussian mixture to the proposal distribution mixture according to its contribution to the product, weighted sample allocation makes the proposal distribution mixture closer to the product, thus improving the efficiency of importance sampling.

The weighted mixture importance sampling approach is summarized in Algorithm 2 for the distributed fusion step.

Recovery step

We implement the recovery step (2.16) in a similar way via weighted mixture importance sampling. Let GM_k be the fully fused posterior held by S_k and $\mathrm{GM}_{\mathrm{p}k}$ be the prior prediction of the current state by S_k. We draw half of the samples from GM_k and the other half from $\mathrm{GM}_{\mathrm{p}k}$. For a sample $\boldsymbol{x}_n^{(m)}$ drawn from GM_k, its importance weight is calculated as

$$w_n^{(m)} = \frac{\mathrm{GM}_k(\boldsymbol{x}_n^{(m)})^K}{\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}_n^{(m)})^{K-1}\mathrm{GM}_k(\boldsymbol{x}_n^{(m)})} = \frac{\mathrm{GM}_k(\boldsymbol{x}_n^{(m)})^{K-1}}{\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}_n^{(m)})^{K-1}}; \tag{2.20}$$

Algorithm 2: GM Fusion

procedure GMFUSE (GM_k, $\{\mathrm{GM}_j\}_{j\in N_k}$)
  initialize M, $\{\varepsilon_{kj}\}_{j\in N_k}$
  for j **in** N_k **do**
    $M_j = \lfloor M\varepsilon_{kj} \rfloor$
    generate $\{\boldsymbol{x}_j^{(m)}\}_{m=1}^{M_j}$ from GM_j
    for $m = 1$ to M_j
      $w_j^{(m)} = \mathrm{GM}_j(\boldsymbol{x}_j^{(m)})^{-1} \prod_{l\in N_k} \mathrm{GM}_l(\boldsymbol{x}_j^{(m)})^{\epsilon_{k,l}}$
    end for
  end for
  normalize $\{w_j^{(m)}\}$
  return GMLEARN($\{\boldsymbol{x}_j^{(m)}, w_j^{(m)}\}$)
end procedure

for a sample $\boldsymbol{x}_n^{(m)}$ drawn from $\mathrm{GM}_{\mathrm{p}k}$, its importance weight is calculated as

$$w_n^{(m)} = \frac{\mathrm{GM}_k(\boldsymbol{x}_n^{(m)})^K}{\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}_n^{(m)})^{K-1}\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}_n^{(m)})} = \frac{\mathrm{GM}_k(\boldsymbol{x}_n^{(m)})^K}{\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}_n^{(m)})^K}. \tag{2.21}$$

A Gaussian mixture model of the recovered global posterior is then learned from the weighted samples $\{\boldsymbol{x}_n^{(m)}, w_n^{(m)}\}$ using Algorithm 1. Note that we do not apply weighted sample allocation to the recovery step, because negative weights are not justified in allocation.

The recovery step is summarized in Algorithm 3.

As we can see, the fusion of Gaussian mixtures in Algorithms 2 and 3 depends only on the density function described by each Gaussian mixture and does not care about how many components each Gaussian mixture has. In other words, the proposed method gives each individual sensor the flexibility to choose an optimal, yet not necessarily uniform, number of components based on its own samples, thus improving approximation accuracy and efficiency.

Algorithm 3: GM Recovery

procedure GMRecover(GM_k, $\mathrm{GM}_{\mathrm{p}k}$)
  initialize M
  generate $\{\boldsymbol{x}^{(m)}\}_{m=1}^{\lfloor M/2 \rfloor}$ from GM_k
  for $m = 1$ to $\lfloor M/2 \rfloor$
    $w^{(m)} = \left[\mathrm{GM}_k(\boldsymbol{x}^{(m)})/\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}^{(m)})\right]^{K-1}$
  end for
  generate $\{\boldsymbol{x}^{(m)}\}_{m=\lfloor M/2 \rfloor+1}^{M}$ from $\mathrm{GM}_{\mathrm{p}k}$
  for $m = \lfloor M/2 \rfloor + 1$ to M
    $w^{(m)} = \left[\mathrm{GM}_k(\boldsymbol{x}^{(m)})/\mathrm{GM}_{\mathrm{p}k}(\boldsymbol{x}^{(m)})\right]^{K}$
  end for
  normalize $\{w^{(m)}\}_{m=1}^{M}$
  return GMLearn($\{\boldsymbol{x}^{(m)}, w^{(m)}\}_{m=1}^{M}$)
end procedure

In comparison, most other posterior-based algorithms fuse local posteriors based on their parameters rather than the density functions described by these parameters, and thus put structural constraints on local parametric representations. For example, linear fusion of Gaussian mixtures [12], [13] requires each mixture to have the same number of components. This requirement gives little flexibility to sensors and compromises adaptivity in local signal processing.

2.4.4 Summary of distributed particle filtering

We summarize the proposed distributed particle filtering algorithm in Algorithm 4, in which "PF" is short for "particle filtering," and the convergence in fusion is locally determined when the discrepancy in belief is lower than a certain threshold under a chosen metric and no neighbor is still sending data. We do not specify the exact particle filter for local particle

Algorithm 4: Distributed Particle Filtering

procedure DPF($\{\boldsymbol{x}_{n-1,k}^{(m)}, w_{n-1,k}^{(m)}\}_{m,k=1}^{M,K}$, $\{\boldsymbol{y}_k\}_{k=1}^{K}$)
  for $k = 1$ to K **in parallel filtering**
    $\{\boldsymbol{x}_{n,k}^{(m)}, w_{n,k}^{(m)}\}_{m=1}^{M}$=PF($\{\boldsymbol{x}_{n-1,k}^{(m)}, w_{n-1,k}^{(m)}\}_{m=1}^{M}, \boldsymbol{y}_{n,k}$)
    $\mathrm{GM}_{\mathrm{p}k}$ = GMLearn($\{\boldsymbol{g}(\boldsymbol{x}_{n-1,k}^{(m)}), w_{n-1,k}^{(m)}\}_{m=1}^{M}$)
    GM_k = GMLearn($\{\boldsymbol{x}_{n,k}^{(m)}, w_{n,k}^{(m)}\}_{m=1}^{M}$)
  end for
  repeat fusion
    for $k = 1$ to K **in parallel**
      S_k sends GM_k to S_j for $\forall j \in N_k$
    end for
    for $k = 1$ to K **in parallel**
      GM_k = GMFuse(GM_k, $\{\mathrm{GM}_j\}_{j \in N_k}$)
    end for
  until convergence
  for $k = 1$ to K **in parallel recovery**
    GM_k = GMRecover(GM_k, $\mathrm{GM}_{\mathrm{p}k}$)
    generate $\{\boldsymbol{x}_{n,k}^{(m)}\}_{m=1}^{M}$ from GM_k
  end for
  return $\{\boldsymbol{x}_{n,k}^{(m)}, 1/M\}_{m,k=1}^{M,K}$
end procedure

filtering, because each sensor can select its own customized particle filter, thanks to the flexibility given by posterior-based fusion.

2.5 Performance analysis

In this section, we investigate the performance of the proposed distributed particle filtering algorithm in terms of convergence, communication overhead, and computational complexity.

2.5.1 Convergence of average consensus

A standard average consensus algorithm is proved to converge under certain conditions in [1], [26]. However, the proof for standard average consensus does not directly apply to the average consensus algorithm proposed in Section 2.4.1, because the proposed algorithm has an additional normalization step (2.13) for each sensor in each iteration and thus is different from standard average consensus. We claim the convergence of the proposed average consensus algorithm in (2.15) and show its convergence below.

Theorem 1. *After a sufficiently large number of iterations of average consensus with normalization, the posterior held by each sensor converges to the normalized geometric mean of the initial local posteriors obtained from local particle filters.*

Proof. The exponential form of (2.12) with normalization (2.13) can be written as

$$\eta_k^{(i+1)}(\boldsymbol{x}_n) = \gamma_k^{(i+1)} \prod_{j \in N_k} \left(\eta_j^{(i)}(\boldsymbol{x}_n) \right)^{\varepsilon_{kj}}, \tag{2.22}$$

where $\gamma_k^{(i+1)}$ is a constant coefficient that normalizes $\eta_k^{(i+1)}(\boldsymbol{x}_n)$ so that it integrates to one. Each consensus iteration involves such a constant coefficient for each sensor, and the constant coefficient accumulates across iterations. We denote the part of $\eta_k^{(i)}$ that comes purely from the fusion of the original posteriors obtained from local particle filters (in other words, $\eta_k^{(0)}$) as $p_k^{(i)}(\boldsymbol{x}_n)$, and the accumulated constant coefficient that $\eta_k^{(i)}$ has collected up to the ith iteration as $\lambda_k^{(i)}$. Then, we have $\eta_k^{(i)}(\boldsymbol{x}_n) = \lambda_k^{(i)} p_k^{(i)}(\boldsymbol{x}_n)$. When $i = 0$, we have $\lambda_k^{(0)} = 1$ and

$p_k^{(0)}(\boldsymbol{x}_n) = f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$; when $i \geq 1$, we have

$$\begin{aligned}\eta_k^{(i+1)}(\boldsymbol{x}_n) &= \gamma_k^{(i+1)} \prod_{j\in N_k} \left(\lambda_j^{(i)} p_j^{(i)}(\boldsymbol{x}_n)\right)^{\varepsilon_{kj}} \\ &= \underbrace{\gamma_k^{(i+1)} \prod_{j\in N_k} \left(\lambda_j^{(i)}\right)^{\varepsilon_{kj}}}_{\lambda_k^{(i+1)}} \times \underbrace{\prod_{j\in N_k} \left(p_j^{(i)}(\boldsymbol{x}_n)\right)^{\varepsilon_{kj}}}_{p_k^{(i+1)}(\boldsymbol{x}_n)}. \end{aligned} \tag{2.23}$$

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

$$\begin{aligned}\log p_k^{(i+1)}(\boldsymbol{x}_n) &= \sum_{j\in N_k} \varepsilon_{kj} \log p_j^{(i)}(\boldsymbol{x}_n) \\ &= \log p_k^{(i)}(\boldsymbol{x}_n) + \sum_{j\in N_k} \varepsilon_{kj} \left(\log p_j^{(i)}(\boldsymbol{x}_n) - \log p_k^{(i)}(\boldsymbol{x}_n)\right), \end{aligned} \tag{2.24}$$

which coincides with the canonical form of (weighted) average consensus. With the underlying graph $\boldsymbol{G}$ being connected and not bipartite, according to [1], [26] we have

$$\lim_{i\to\infty} \log p_k^{(i)}(\boldsymbol{x}_n) = \frac{1}{K} \sum_{k=1}^{K} \log p_k^{(0)}(\boldsymbol{x}_n), \tag{2.25}$$

or equivalently,

$$\begin{aligned}\lim_{i\to\infty} p_k^{(i)}(\boldsymbol{x}_n) &= \prod_{k=1}^{K} \left(p_k^{(0)}(\boldsymbol{x}_n)\right)^{\frac{1}{K}} \\ &= \prod_{k=1}^{K} f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})^{\frac{1}{K}}. \end{aligned} \tag{2.26}$$

Hence,

$$\begin{aligned}\lim_{i\to\infty}\eta_k^{(i)}(\boldsymbol{x}_n) &= \lim_{i\to\infty}\lambda_k^{(i)}p_k^{(i)}(\boldsymbol{x}_n)\\ &= \lim_{i\to\infty}\lambda_k^{(i)}\lim_{i\to\infty}p_k^{(i)}(\boldsymbol{x}_n)\\ &= \Big(\lim_{i\to\infty}\lambda_k^{(i)}\Big)\prod_{k=1}^{K}f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k},\boldsymbol{y}_{1:n-1})^{\frac{1}{K}},\end{aligned} \tag{2.27}$$

where $\lim_{i\to\infty}\eta_k^{(i)}(\boldsymbol{x}_n)$ is the posterior held by S_k at convergence, $\prod_{k=1}^{K}f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k},\boldsymbol{y}_{1:n-1})^{\frac{1}{K}}$ is the geometric mean of the initial local posteriors, and $\lim_{i\to\infty}\lambda_k^{(i)}$ normalizes the geometric mean so that it exists as a valid probability density function in the form of $\lim_{i\to\infty}\eta_k^{(i)}(\boldsymbol{x}_n)$.

□

Following the convergence, the global posterior can be obtained separately by each sensor through a recovery step, which results from substituting (2.27) into (2.10).

2.5.2 Convergence of distributed particle filtering

The proposed distributed particle filtering algorithm implements the proposed average consensus algorithm with approximations and asymptotically converges under the following three 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 these can be perfectly satisfied without considerable communication or computation. Hence, convergence errors are usually inevitable. Due to independent randomness, different sensors are likely to have

different convergence errors, thus resulting in consensus errors. Although the proposed algorithm does not require exact consensus as weight-based algorithms do, inexact consensus, if too significant, can lead to future errors in both filtering and fusion.

Consensus errors can be manually eliminated by additional average consensus on the parameters of the obtained Gaussian mixture models [13]. As mentioned in the Introduction, parameter-based average consensus is not justified by the underlying statistical model and thus is suboptimal in the fusion of local posteriors. However, its suboptimality is not a problem here, because the method is used not for fusion but for numerical fine-tuning of beliefs that are already close to consensus. Also, because of the closeness to consensus, it is not expected to take many consensus iterations.

Note that parameter-based average consensus requires that all the Gaussian mixtures to be fused have the same number of components. To satisfy the constraint, we have to adjust the number of components for each Gaussian mixture in case they do not agree. We achieve this via sampling. More specifically, we first sample from each Gaussian mixture and then learn from the samples a Gaussian mixture model with a specified uniform number of components.

2.5.3 Communication overhead

In the proposed algorithm, posteriors are transmitted between sensors in the form of Gaussian mixtures. Let C be the average number of components in these Gaussian mixtures, then we need to transmit $C(d^2 + d + 1)$ numbers per Gaussian mixture, with d being the state dimension. Since covariance matrices are symmetric, we only need to transmit $(d^2 + d)/2$, instead of d^2, numbers for each covariance matrix in a Gaussian mixture. Also, since component weights sum to one, we only need to transmit $C - 1$, instead of C, component

weights. Thus, the actual count of numbers needed to represent a Gaussian mixture is $Cd^2/2 + (C/2+1)d + C - 1$. In a consensus iteration, each communication link is used once in each direction, so the total number of Gaussian mixtures transmitted in a consensus iteration is $2|\boldsymbol{E}|$. Let L be the number of consensus iterations, and then the proposed algorithm communicates $2|\boldsymbol{E}|L(Cd^2/2 + (C/2+1)d + C - 1)$ numbers in total and $2|\boldsymbol{E}|L(Cd^2/2 + (C/2+1)d + C - 1)/K$ numbers per sensor during each time step. Since $|\boldsymbol{E}|$ ranges from $O(K)$ to $O(K^2)$ for a connected graph, the communication complexity per sensor is between $O(LCd^2)$ and $O(KLCd^2)$.

In comparison, the count of numbers transmitted by each sensor in a weight-based algorithm is proportional to the number of particles, whose is proved to grow exponentially with the state dimensionality d for a successful particle filter [34]; the communication complexity of a likelihood-based algorithm is combinatorial in the state dimension, because the number of regression coefficients needed for the polynomial approximation presented in [8] is combinatorial with the state dimension; the communication complexity of a posterior-based algorithm with the Gaussian approximations is quadratic in the state dimension, with d and $(d^2+d)/2$ numbers to represent the mean and covariance matrix, respectively, of a Gaussian distribution.

It is hard to directly compare the communication complexity of the proposed algorithm with those of other algorithms, because the dependence of L and C on d is mostly problem-specific. In Section 2.6.5, we compare the actual communication costs of different algorithms through numerical examples.

2.5.4 Computational complexity

We now investigate the computational complexity of distributed particle filtering algorithms, focusing on the fusion part without considering the filtering part, because the latter has almost the same complexity among different algorithms. Since a distributed particle filtering algorithm runs at each sensor in parallel, we only consider the computation performed at a single sensor.

The proposed algorithm calls Algorithms 1, 2, and 3. Algorithm 1 (GM learning) costs $O(L_g M_g C d^2)$ to learn a Gaussian mixture of C components from M_g samples in L_g iterations. Algorithm 2 (GM fusion) calls Algorithm 1 once and costs $O(K M_f C d^2)$ in addition to calling Algorithm 1, where K means that a sensor has at most $O(K)$ neighbors and M_f is the sample size for importance sampling in distributed fusion. Algorithm 3 (GM recovery) also calls Algorithm 1 once and costs $O(M_r C d^2)$ in addition to calling Algorithm 1, with M_r being the sample size for importance sampling in recovery. In addition to Algorithms 1, 2, and 3, the proposed algorithm calls the parameter-based average consensus algorithm for numerical fine-tuning, which costs $O(KCd^2)$ in each iteration. In summary, if we assume that the proposed algorithm takes L_f iterations for distributed fusion and L_p iterations for fine-tuning, then the overall computational complexity is $O((L_f L_g M_g + K L_f M_f + M_r + K L_p) C d^2)$. Assuming that M_g, M_f, and M_r are all $O(M)$, then the complexity simplifies to $O([(L_g + K) L_f M + K L_p] C d^2)$.

In comparison, a likelihood-based algorithm [8] costs $O(R^3 + (M + q)R^2 + (d + q)MR)$ on polynomial approximation (M is the sample size, q is the dimension of the state function appearing in factorization, and R is the dimension of the polynomial basis expansion) and

$O(LKR)$ on consensus (L is the number of consensus iterations). Thus, the overall complexity is $O(R^3+(M+q)R^2+[(d+q)M+LK]R)$. Since R itself is a combinatorial function of the state dimension d, the cubic function of R might make the algorithm scale poorly in high-dimensional systems. A weight-based algorithm [6] only needs to perform average consensus on weights with a computational complexity of $O(LKM)$. A Gaussian posterior-based algorithm costs $O(LKd^3)$. Generally, the proposed algorithm and the likelihood-based algorithm require more computation than the weight-based algorithm and the Gaussian posterior-based algorithm. The former two algorithms use a certain compact representation for inter-sensor communication and thus need to enclose information in and read information out of the representation. Such a representation incurs much lower communication overhead than particles and provides a more accurate approximation than a single Gaussian distribution, as shown in Section 2.6.

2.6 Numerical examples

In this section, we demonstrate the performance of the proposed distributed particle filtering algorithm in comparison with weight-based, likelihood-based, and other posterior-based algorithms, through numerical examples of decentralized target tracking.

2.6.1 General settings

We considered a wireless sensor network consisting of 20 sensors programmed to track a moving target.

The target followed a Wiener process acceleration model [35] in two-dimensional space. The target state consisted of the position, velocity, and acceleration of the target along each dimension as

$$\boldsymbol{x}_n = \begin{bmatrix} x_{n,1} & x_{n,2} & \dot{x}_{n,1} & \dot{x}_{n,2} & \ddot{x}_{n,1} & \ddot{x}_{n,2} \end{bmatrix}^T \tag{2.28}$$

The state transition function was

$$\boldsymbol{g}(\boldsymbol{x}_n) = \boldsymbol{D} \cdot \boldsymbol{x}_n, \tag{2.29}$$

where

$$\boldsymbol{D} = \begin{bmatrix} 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 \\ 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 \\ 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

with t being the state transition interval. The state transition noise $\boldsymbol{u}_n$ followed a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$, where

$$\boldsymbol{R} = \sigma_u^2 \begin{bmatrix} \frac{1}{20}t^5 & 0 & \frac{1}{8}t^4 & 0 & \frac{1}{6}t^3 & 0 \\ 0 & \frac{1}{20}t^5 & 0 & \frac{1}{8}t^4 & 0 & \frac{1}{6}t^3 \\ \frac{1}{8}t^4 & 0 & \frac{1}{3}t^3 & 0 & \frac{1}{2}t^2 & 0 \\ 0 & \frac{1}{8}t^4 & 0 & \frac{1}{3}t^3 & 0 & \frac{1}{2}t^2 \\ \frac{1}{6}t^3 & 0 & \frac{1}{2}t^2 & 0 & t & 0 \\ 0 & \frac{1}{6}t^3 & 0 & \frac{1}{2}t^2 & 0 & t \end{bmatrix}. \tag{2.31}$$

We assumed that the target traveled over 30 unit-length state transition intervals.

Each sensor measured the range and range rate (Doppler) of the target. The kth sensor, S_k, was located at $\boldsymbol{l}_k = (l_{k,1}, l_{k,2})$ with the observation function

$$\boldsymbol{h}_k(\boldsymbol{x}_n) = \begin{bmatrix} h_{k,\text{range}}(\boldsymbol{x}_n) & h_{k,\text{doppler}}(\boldsymbol{x}_n) \end{bmatrix}^T, \tag{2.32}$$

where

$$h_{k,\text{range}}(\boldsymbol{x}_n) = \sqrt{(x_{n,1} - l_{k,1})^2 + (x_{n,2} - l_{k,2})^2} \tag{2.33}$$

and

$$h_{k,\text{doppler}}(\boldsymbol{x}_n) = \frac{\dot{x}_{n,1}(x_{n,1} - l_{k,1}) + \dot{x}_{n,2}(x_{n,2} - l_{k,2})}{\sqrt{(x_{n,1} - l_{k,1})^2 + (x_{n,2} - l_{k,2})^2}}, \tag{2.34}$$

and the observation noise

$$\boldsymbol{v}_{n,k} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_w^2 \end{bmatrix} \right). \tag{2.35}$$

We set σ_u as 0.5, σ_v as 1, and σ_w as 1. We set the neighborhood radius threshold ρ according to the sensor locations to ensure that the network was connected. The initial target state $\boldsymbol{x}_0$ was set as $\mathbf{0}$, and $\mathcal{N}(\boldsymbol{x}_0, \boldsymbol{R})$ was assumed as the prior information available to each sensor. In a predefined fashion, we assumed three components for any Gaussian mixture. We used the sampling importance resampling (SIR) particle filter [36] for local particle filtering. We used 20,000 samples for importance sampling in both distributed fusion and recovery of the proposed algorithm. We tested all the algorithms on multiple sensor networks and randomly generated trajectories, and compared the average performance. Fig. 2.1 shows an example of a sensor network with a target trajectory randomly generated under the Wiener process acceleration model.

We compared the proposed algorithm ("Optimal GM") with other posterior-based algorithms, including the Bayesian fusion of Gaussian approximations ("Bayesian Gauss") [10]

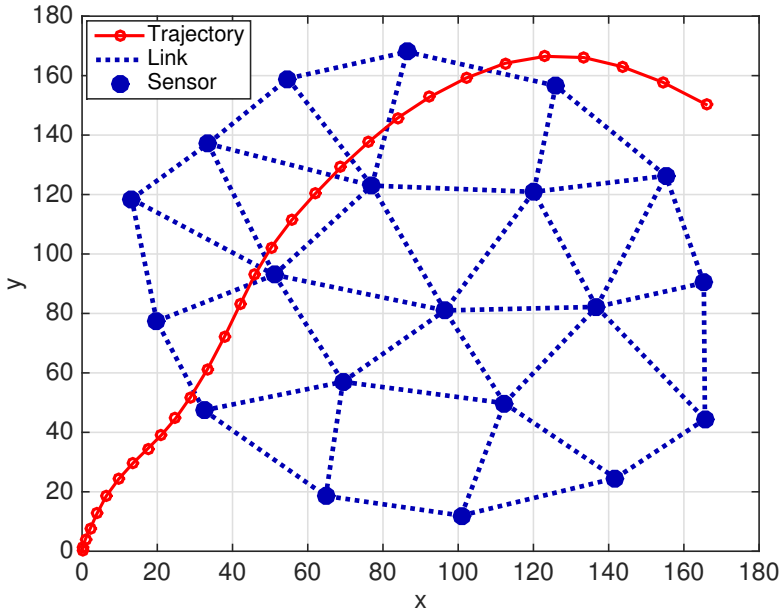


Figure 2.1: An example of a wireless sensor network, its local communication links, and a target trajectory.

and the linear fusion of Gaussian mixtures ("Linear GM") [13]. We also compared it with a representative weight-based algorithm [6], likelihood-based algorithm [8], and distributed unscented Kalman filter (UKF) [37], which can be also considered as a posterior-based algorithm, although it does not involve particle filtering. Moreover, we compared it with centralized particle filtering, which served as a benchmark.

2.6.2 Metrics

We considered the posterior mean as a point estimate of each state and used the root-mean-square error (RMSE) to quantify the performance. For a single state $\boldsymbol{x}_n$, the RMSE of an estimate $\hat{\boldsymbol{x}}_n$ was defined as $||\hat{\boldsymbol{x}}_n - \boldsymbol{x}_n||$, namely the l-2 norm of $\hat{\boldsymbol{x}}_n - \boldsymbol{x}_n$; for a state sequence of length T, i.e., $\{\boldsymbol{x}_n\}_{n=1}^T$, the average RMSE (ARMSE) of a sequence estimate, $\{\hat{\boldsymbol{x}}_n\}_{n=1}^T$,

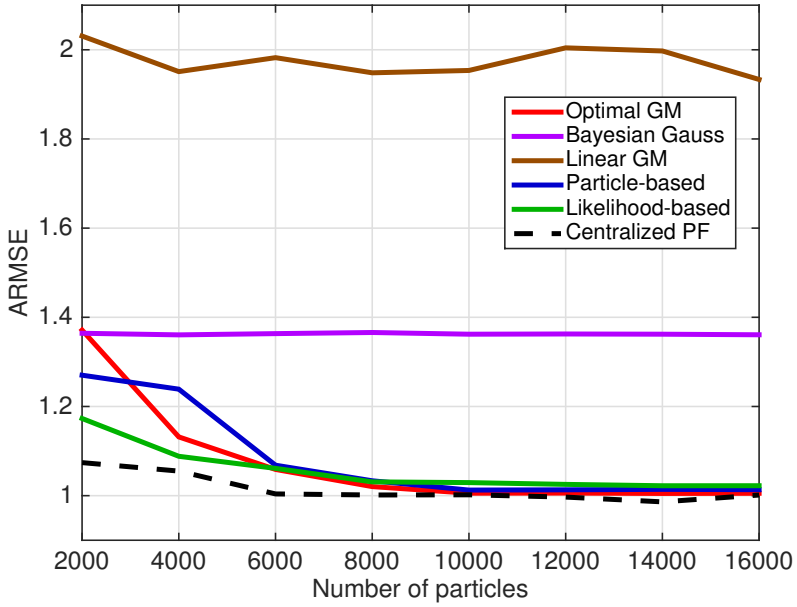


Figure 2.2: Trajectory estimation ARMSE as a function of the number of particles

was defined as $\sqrt{\frac{1}{T}\sum_{n=1}^{T}||\hat{\boldsymbol{x}}_n - \boldsymbol{x}_n||^2}$. In a network that performs distributed filtering, each sensor holds a separate global estimate and thus has its own RMSE and ARMSE. We used their averages to quantify the performance of the whole network. We used the Kullback-Leibler (KL) distance [38] to describe the dissimilarity between two Gaussian mixtures. Since it is analytically intractable to compute the KL distance between two Gaussian mixtures, we approximated it using the Gaussian approximation approach introduced in [39].

2.6.3 Accuracy

We tested all the methods to be investigated on multiple examples to compare their average trajectory estimation accuracy. Fig. 2.2 compares the ARMSEs as a function of the number of particles, given sufficient consensus iterations. We can see that the error of the proposed method varied the most with the number of particles. With 2,000 particles, its error was

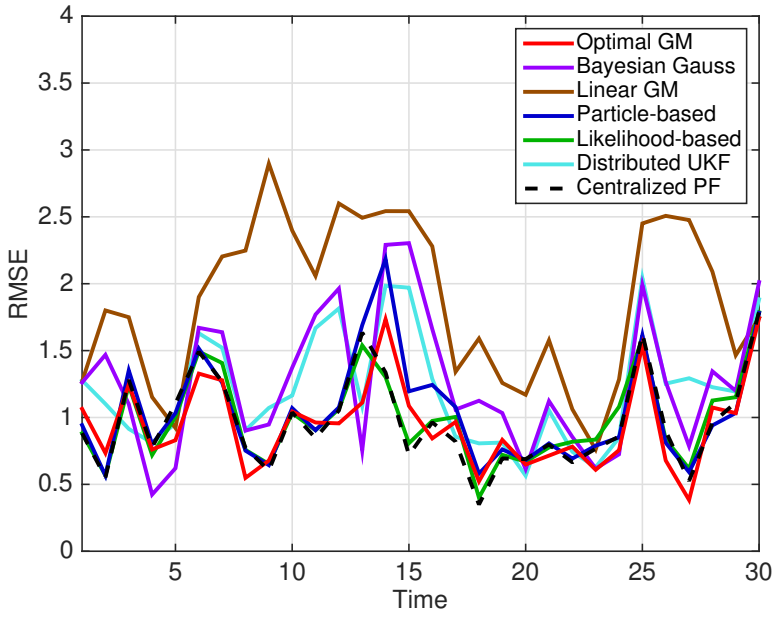


Figure 2.3: State estimation RMSE as a function of time.

lower than that of Linear GM only; with no less than 8,000 particles, its error was close to that of centralized particle filtering and no higher than that of any other method. The performance of the proposed method varied significantly because the approximation accuracy of a Gaussian mixture is strongly affected by the number of particles used in local particle filtering. In contrast, the error of Bayesian Gauss, also a posterior-based method, stayed almost constant with different numbers of particles, because the accuracy of a Gaussian approximation, including a mean and a covariance matrix only, is relatively robust to the number of particles used to represent a local posterior. The error of Linear GM, another posterior-based method, did not vary much with the number of particles either, because Linear GM failed to benefit from the increased number of particles due to its unjustified fusion rule. The errors of both the likelihood-based and weight-based methods dropped as the number of particles increased. Their errors were lower than that of the proposed method when we had a small number of particles, and comparable to that of the proposed

method when we had a medium or large number of particles. In summary, the proposed method was the most accurate among all the posterior-based methods and competitive with the likelihood-based and weight-based methods, when the number of particles was not too small.

We also tested all the methods on the example in Fig. 2.1 to investigate their state estimation accuracy. For each method, we used the number of particles corresponding to its elbow point in Fig. 2.2, i.e., 10,000 particles for the proposed method, 2,000 for Bayesian Gauss, 4,000 for Linear GM, 10,000 for the weight-based method, 8,000 for the likelihood-based method, and 6,000 for centralized particle filtering. In Fig. 2.3, we show the state estimation RMSE of each method as a function of time along the trajectory of the target. We can see that the proposed method, the likelihood-based method, the weight-based method, and centralized particle filtering had state estimation errors at almost the same level, while Linear GM, Bayesian Gauss, and distributed UKF suffered from high errors at many time points. Among all the methods, Linear GM obviously yielded the highest errors, which again demonstrates the deficiency of the unjustified linear fusion rule.

2.6.4 Consensus

We investigated the consensus process of the proposed method within a single time step. Fig. 2.4 shows the consensus process during the 10th time step as an example, in which we applied the proposed method, with 10,000 particles for local filtering, 20 iterations for average consensus, and 20 iterations for numerical fine-tuning, to the example in Fig. 2.1. We can see that in both average consensus and numerical fine-tuning, both the KL distance and the RMSE dropped and converged as the algorithm proceeded, which demonstrated the

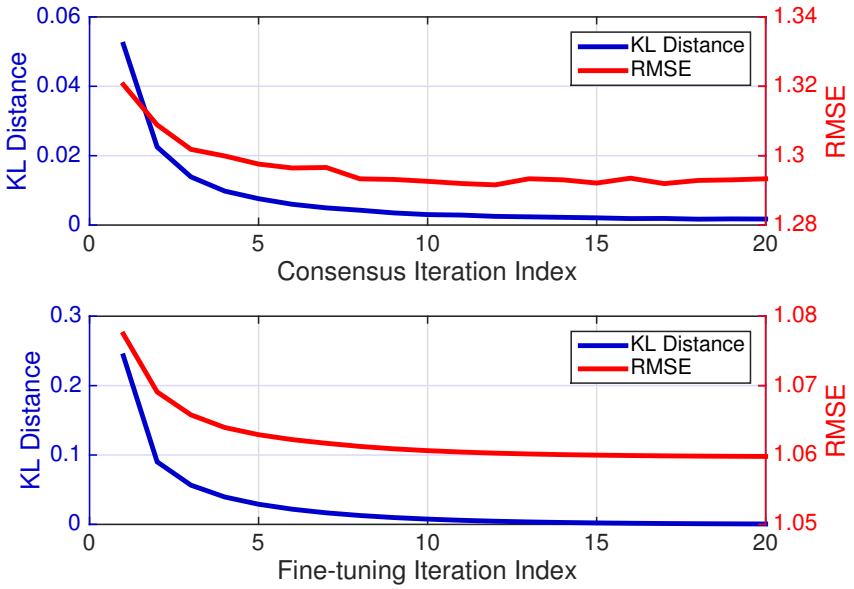


Figure 2.4: KL distance and state estimation RMSE across iterations during the 10th time step using the proposed method

validity of the proposed average consensus algorithm in terms of convergence. Note that the metrics in Fig. 2.4 were computed based on unrecovered beliefs for average consensus and recovered beliefs for numerical fine-tuning, so they came in different scales.

2.6.5 Communication overhead

We investigated the communication overhead of each method and the relationship between communication overhead and estimation accuracy. For each method, we fixed the number of particles at its elbow point, as specified in Section 2.6.3, and investigated its performance with the number consensus iterations varying.

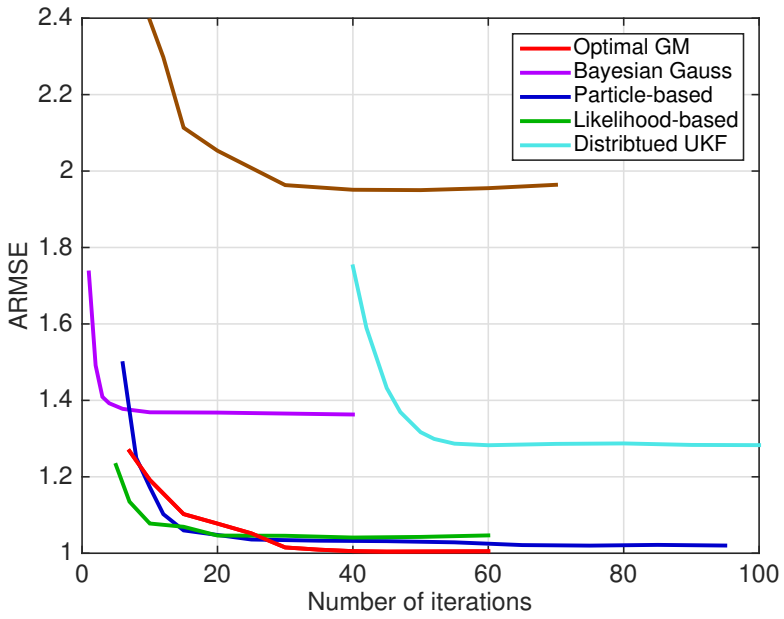


Figure 2.5: Trajectory estimation ARMSE as a function of the number of consensus iterations

In Fig. 2.5, we demonstrate the effect of the number of consensus iterations on the performance of a distributed filtering algorithm. As we can see, the error of each method dropped as the number of iterations increased and stayed constant beyond a certain threshold.

In Fig. 2.6, we show the trajectory estimation ARMSE of each method as a function of the communication overhead per time step. We used the count of numbers transmitted between sensors in the network to quantify the communication overhead of each method. As expected, there was a trade-off between estimation accuracy and communication efficiency. For each method, the estimation error dropped as the communication overhead increased, but stayed almost constant beyond a certain threshold. The proposed method, the weight-based method, and the likelihood-based method had errors at the same level but communication costs of different orders of magnitude. The weight-based method, which communicated non-parametric approximations, transmitted more numbers than the proposed method and

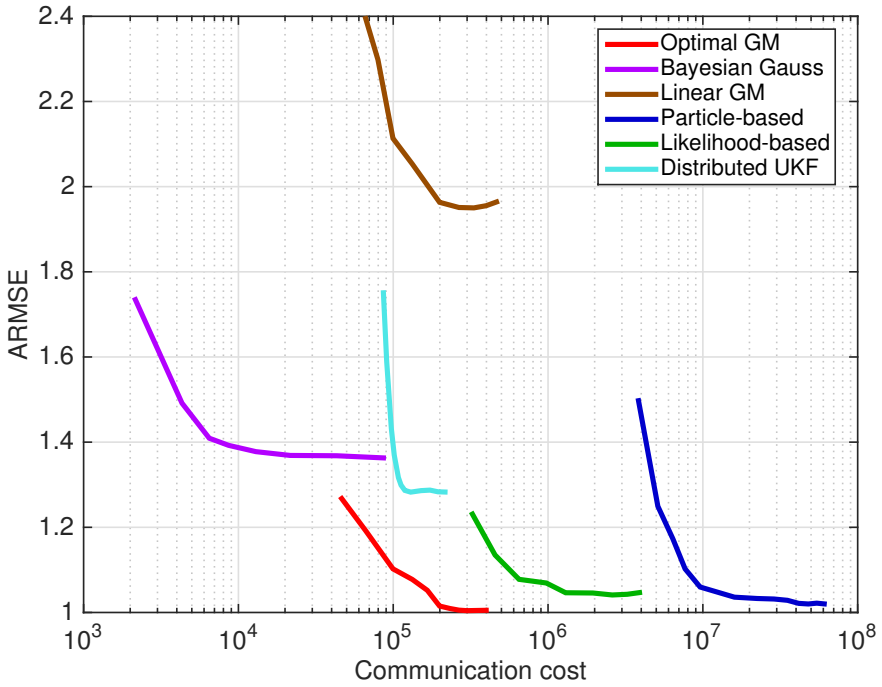


Figure 2.6: Trajectory estimation ARMSE as a function of the communication cost per time step

the likelihood-based method, both of which communicated parametric approximations. The likelihood-based method, which used polynomial approximations, transmitted more numbers than the proposed method, which used Gaussian mixture approximations. Bayesian Gauss and distributed UKF, both posterior-based methods, had errors at the same level, higher than that of the proposed method, due to the insufficient approximation accuracy of Gaussian approximations. The communication overhead of distributed UKF was close to that of the proposed method, while that of Bayesian Gauss was lower than that of any other method in Fig. 2.6. Note that the trade-off between estimation accuracy and communication efficiency existed not only within each method, but also between different methods. As we can see, the proposed method was more accurate than Bayesian Gauss, benefiting from the upgrade from Gaussian approximations to Gaussian mixture approximations, but in the meantime incurred

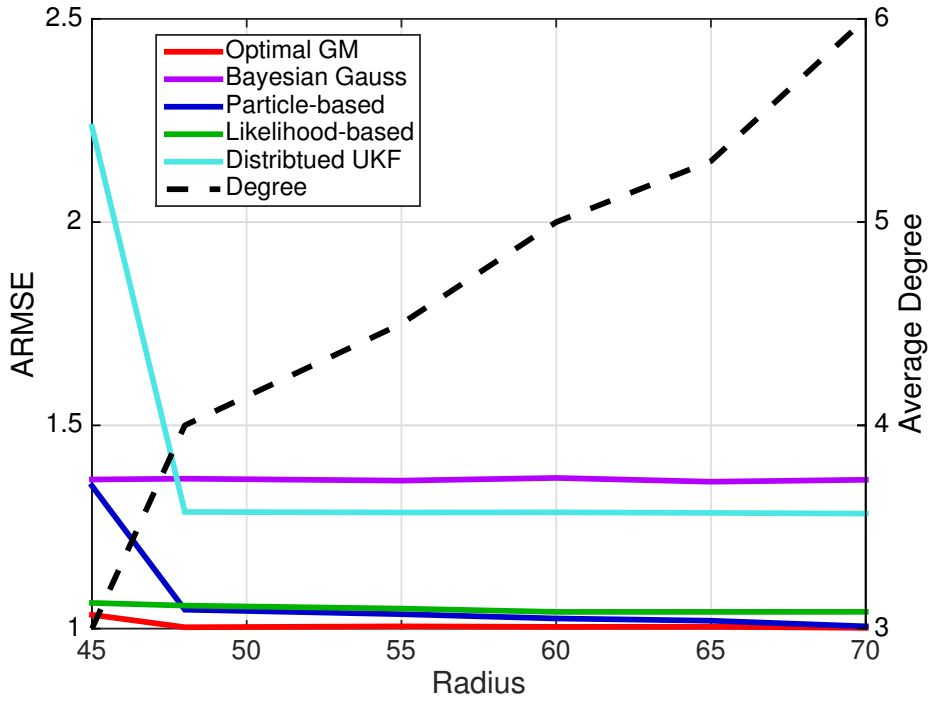


Figure 2.7: Trajectory estimation ARMSE and average degree as functions of the local communication radius

extra communication overhead due to the upgrade. Given the significant improvement in accuracy, we claim that the extra communication incurred by the Gaussian mixture model used in the proposed method was justified.

2.6.6 Local communication radius

The local communication radius determines the neighborhood and the number of neighbors for each sensor. Fig. 2.7 shows the effect of the radius on the performance of distributed particle filtering methods, with both the number of particles and the number of consensus iterations fixed at the respective elbow points corresponding to each method. The simulations were conducted on the network in Fig. 2.1, whose default radius was 48. As we can see in Fig.

2.7, when the radius was lower than the default radius, the errors of distributed UKF and weight-based method increased dramatically, and those of the proposed method, Bayesian Gauss, and the likelihood-based method increased slightly; when the radius was higher than the default radius, the error of each method either stayed constant or decreased slightly. In fact, the radius controls the rate of consensus. When the radius is small, it might takes many iterations of communication for information to be transmitted from a sensor to another in the network; when the radius is sufficiently large, a sensor can communicate directly with any other sensor in the network, and the network becomes equivalently centralized. When the number of consensus iterations is fixed, the radius effectively controls the progress of consensus. Thus, when a radius is large enough for the network to reach consensus within the given number of consensus iterations, it would not help much to further increase the radius, as shown in Fig. 2.7. Also, since a large radius adds to the difficulty in communication, it might not be always desirable to increase the radius in distributed fusion.

2.7 Chapter summary

In this chapter, we proposed a distributed particle filtering algorithm based on optimal fusion of local posteriors approximated as Gaussian mixtures. We implemented the optimal fusion rule in a distributed fashion via an average consensus algorithm. We derived a distributed fusion rule for the consensus algorithm and performed the fusion of Gaussian mixtures via importance sampling. With an extra normalization step involved in the distributed fusion rule, the convergence of the proposed average consensus algorithm does not directly follow that of a standard average consensus algorithm. We therefore proved the convergence of the proposed average consensus algorithm and then validated it with numerical examples. We

also demonstrated the performance of the proposed distributed particle filtering algorithm through numerical examples. The numerical examples showed that the error of the proposed algorithm was at least 27% lower than that of the other posterior algorithms and slightly lower than those of the particle-based and posterior-based algorithms, which implies that the proposed algorithm significantly improves the accuracy of posterior-based algorithms and is competitive in accuracy with state-of-the-art approaches. The numerical examples also showed that the proposed algorithm incurred a communication cost that was 1% that of the particle-based algorithm and 10% that of the likelihood-based algorithm, which implies that the proposed algorithm is efficient in communication. The numerical examples further showed that the posterior-based algorithm using Gaussian approximations incurred a communication cost that was 10% that of the proposed algorithm and achieved an estimation error that was 37% higher than that of the proposed algorithm, which implies a trade-off between communication efficiency and approximation accuracy. We claim that the extra communication cost incurred by the upgrade from a Gaussian approximation to a Gaussian mixture model is justified by the increase in the estimation accuracy.

The advantages of the proposed distributed particle filtering algorithm extend beyond accuracy and communication efficiency. As a posterior-based algorithm, it allows diverse sensing modalities and filtering tools to be exploited by the network; by performing importance sampling for nonlinear fusion, it gives each sensor the flexibility to choose the optimal Gaussian mixture model to represent its local belief.

The proposed distributed particle filtering framework has a wide range of applications in addition to target tracking. For example, the distributed particle filtering algorithm can be used in environmental monitoring, smart grids, and situational awareness; the distributed fusion rule can be also applied to general nonlinear fusion problems.

Chapter 3

Adaptive Gaussian Mixture Learning

In this chapter, we consider the problem of adaptive Gaussian mixture learning in distributed particle filtering where posteriors are approximated as Gaussian mixtures for wireless communication. We propose a hierarchical clustering algorithm, combined with the EM algorithm, to learn from weighted samples a Gaussian mixture consisting of an adaptively determined number of components. Different from existing work, the proposed algorithm embeds a clustering algorithm based on kernel density estimation in each recursive step of hierarchical clustering to split each cluster in an adaptive fashion. Numerical examples show that the proposed method leads to higher accuracy in distributed particle filtering and is more efficient in both computation and communication than other Gaussian mixture learning methods. [2]

3.1 Introduction

In posterior-based distributed particle filtering, we often parametrically represent a posterior as a Gaussian mixture [14] for wireless transmission. When we fit a Gaussian mixture model

[2]This chapter is based on J. Li and A. Nehorai, "Adaptive Gaussian mixture learning in distributed particle filtering," in *6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, Cancun, Mexico, Dec. 2015, pp. 221–224.

to a posterior or samples drawn from the posterior, an important consideration is how many components the mixture model should have. Generally, the more components we have, the more flexibility we gain in the parametric representation, and the higher approximation accuracy we can achieve. However, the cost of a large number of components is the increased computational load to learn a Gaussian mixture and the increased communication overhead to transmit the Gaussian mixture. For this reason, there is always a trade-off between approximation accuracy and communication efficiency when we decide the number of mixture components we use to approximate a posterior.

Most posterior-based algorithms [10]–[13] assume a uniform number of components in each Gaussian mixture (a Gaussian distribution is a one-component Gaussian mixture) across time and space for convenience in fusion, but the assumption is usually invalid. First, sensors located at different geological spots, with uncorrelated observation noise and possibly different sensing modalities can have different posterior estimates, which are not guaranteed to be accurately described by a mixture of the same number of Gaussian components. Second, as the sensor network approaches consensus, the estimate held by each sensor tends to be more concentrated and thus in general needs fewer Gaussian components to describe. Moreover, even if the noise is assumed to be Gaussian, nonlinearity in the dynamic model still results in non-Gaussian and possibly multimodal posteriors, and thus the Gaussian assumption is not valid. Considering these facts, it would be preferable not to predefine a fixed number of mixture components but rather to determine the optimal number adaptively based on the local data available to each sensor at each time step. We call this approach adaptive Gaussian mixture learning.

A Gaussian mixture is often learned from samples by using the expectation-maximization (EM) algorithm [27]. The EM algorithm is an iterative algorithm that starts from an initial

guess. Since the number of components to use has to be specified in the initial guess, the EM algorithm itself is not adaptive. A possible remedy is to try every possible number of components within a certain range and select the optimal Gaussian mixture model according to a certain model selection criterion, e.g., the Akaike information criterion (AIC) [40] or the Bayesian information criterion (BIC) [41]. Although an adaptive result is guaranteed, this solution is usually very slow due to the heavy computation involved in calling the EM algorithm multiple times. As an iterative algorithm, the EM algorithm itself would be very slow without informed initialization, especially for high-dimensional systems. For this reason, a tree-based hierarchical approach is proposed as a fast and adaptive alternative. In a tree-based approach, the sample set is recursively split into two complementary subsets along a certain dimension, which can be the most variable dimension of the system state, corresponding to a k-dimensional (KD) tree [42], or the principal component of the data, corresponding to a principal component partitioning (PCP) tree [43]; the chosen dimension is often split at the mean or median. After the tree is built, a greedy search is applied to the tree to find the optimal set of components according to a certain objective function. Unlike the EM algorithm, a tree-based approach is not guaranteed to give a local maximum likelihood solution, and thus is subject to inaccuracy.

For Gaussian mixture learning in posterior-based distributed particle filtering, we need efficiency in both computation and communication, in addition to approximation accuracy. As an online algorithm widely used in real-time target tracking, distributed particle filtering requires fast Gaussian mixture learning. Also, since distributed particle filtering is often implemented on a wireless sensor network, which has a tight budget for energy consumption, it also requires each Gaussian mixture to have an appropriate number of components, so that no energy is wasted in transmitting redundant or unnecessary mixture components. For these reasons, no existing methods perfectly apply to our problem.

In this chapter, we propose a new method for adaptive Gaussian mixture learning, based on a combination of hierarchical clustering and the EM algorithm. We design an adaptive splitting strategy for hierarchical clustering to divide the sample set into potential Gaussian components, thus achieving adaptivity, and we then set the output of hierarchical clustering as an initial guess for the EM algorithm, thus achieving accuracy. Thanks to the informed initialization provided by hierarchical clustering based on adaptive splitting, the EM algorithm does not need many iterations to converge, thus achieving efficiency. Based on the proposed method, we propose the first posterior-based distributed particle filtering algorithm equipped with adaptive Gaussian mixture learning.

The rest of the chapter is organized as follows. Section 3.2 describes the signal model. Section 3.3 introduces distributed particle filtering. Section 3.4 proposes a hierarchical clustering algorithm based on adaptive splitting. Section 3.5 validates the proposed method on numerical examples, and Section 3.6 concludes the chapter.

3.2 Signal model

We consider a network of sensors that simultaneously observe a common moving target. We denote the total number of sensors as K and the K sensors as $S_1, S_2, \ldots, S_K$. We assume the agent network to be synchronized [24] and connected. We connect the target activity with the agent network through the following discrete-time state-space model:

$$\begin{cases} \boldsymbol{x}_n = \boldsymbol{g}(\boldsymbol{x}_{n-1}) + \boldsymbol{u}_n \\ \boldsymbol{y}_{n,k} = \boldsymbol{h}_k(\boldsymbol{x}_n) + \boldsymbol{v}_{n,k} \end{cases}, \tag{3.1}$$

where $\boldsymbol{x}_n \in \mathbb{R}^d$ is the target state at the nth time point, $\boldsymbol{y}_{n,k} \in \mathbb{R}^{b_k}$ is the observation of $\boldsymbol{x}_n$ taken by S_k, $\boldsymbol{g}$ is a known state transition function, $\boldsymbol{h}_k$ is a known observation function of S_k, and $\{\boldsymbol{u}_n\}$ and $\{\boldsymbol{v}_{n,k}\}$ are uncorrelated additive noises. For simplicity, we denote $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ as $\boldsymbol{x}_{1:n}$, $\{\boldsymbol{y}_{n,1}, \boldsymbol{y}_{n,2}, \ldots, \boldsymbol{y}_{n,K}\}$ as $\boldsymbol{y}_n$, and $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ as $\boldsymbol{y}_{1:n}$.

3.3 Distributed particle filtering

Distributed particle filtering computes $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$ based on $f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})$ and $\boldsymbol{y}_n$ in a decentralized fashion. Posterior-based distributed particle filtering achieves this goal in two steps. First, each agent S_k computes its local posterior $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$ based on $f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})$ and $\boldsymbol{y}_{n,k}$ through a local particle filter; then, each agent repeatedly communicates with its neighbors and updates its own posterior until they reach consensus on the global posterior $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$.

3.3.1 Local particle filtering

We use the sampling importance resampling (SIR) particle filter [36] for local processing. A SIR particle filter approximates a posterior as a set of weighted samples or particles:

$$f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1}) \approx \sum_{m=1}^{M} w_{n-1}^{(m)} \delta(\boldsymbol{x}_{n-1} - \boldsymbol{x}_{n-1}^{(m)}), \tag{3.2}$$

where M is the total number of particles, $\boldsymbol{x}_{n-1}^{(m)}$ is the mth particle of $\boldsymbol{x}_{n-1}$, $w_{n-1}^{(m)}$ is the normalized weight of $\boldsymbol{x}_{n-1}^{(m)}$, and δ is the Dirac delta function. Then, it propagates each particle from time $n-1$ to time n by sampling from a proposal distribution $f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}^{(m)})$, and

computes the weight of each propagated particle $\boldsymbol{x}_n^{(m)}$ according to

$$w_n^{(m)} \propto w_{n-1}^{(m)} \times f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n^{(m)}). \tag{3.3}$$

The weighted particles $\{\boldsymbol{x}_n^{(m)}, w_n^{(m)}\}$ are resampled if necessary and then considered as a discrete approximation of $f(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$, the local posterior of $\boldsymbol{x}_n$ obtained by S_k.

3.3.2 Distributed fusion

To be compatible with adaptive Gaussian mixture learning, we fuse local posteriors according to a distributed and iterative fusion rule based on importance sampling [44].

The fusion rule starts with an average consensus step, in which every agent iteratively updates its own belief with a weighted average of beliefs in its neighborhood:

$$\log \eta_k^{(i+1)}(\boldsymbol{x}_n) = \sum_{j \in N_k} \varepsilon_{kj} \log \eta_j^{(i)}(\boldsymbol{x}_n), \tag{3.4}$$

where $\eta_j^{(i)}(\boldsymbol{x}_n)$ is the posterior held by A_j in the ith consensus iteration with $\eta_j^{(0)}(\boldsymbol{x}_n)$ being the local posterior obtained by A_j, ε_{kj} is the Metropolis weight [26], and N_k is the neighborhood of S_k with S_k included. The average consensus step terminates when the discrepancy among beliefs is lower than a chosen threshold, and is followed by a recovery step that converts the the consensus result into a global posterior:

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \propto \frac{\eta_k^{(\infty)}(\boldsymbol{x}_n)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}}. \tag{3.5}$$

We approximate the posteriors in (3.4) and (3.5) as Gaussian mixtures and fuse them via importance sampling. Importance sampling only needs the density of each Gaussian mixture evaluated at each given sample and does not care about the number of components in each Gaussian mixture. For this reason, each agent is free to optimize its own approximation through adaptive Gaussian mixture learning.

3.4 Adaptive Gaussian mixture learning

To adaptively approximate posteriors as Gaussian mixtures, we propose an adaptive Gaussian mixture learning method that combines hierarchical clustering with the EM algorithm by setting the output of the former as the initial guess for the latter. To make the combination both accurate and efficient, we develop a hierarchical clustering algorithm with adaptive splitting embedded in each recursive step of tree building.

3.4.1 Dimension reduction

Tree building recursively splits a cluster into two until a termination criterion is satisfied. Since it is often challenging to split a cluster of high-dimensional samples, existing work [45], [43] projects the samples onto a selected dimension before further analysis. In this chapter, we use principal component analysis for dimension reduction because it is able to handle possible correlations between dimensions. More specifically, we apply weighted principal component analysis [46] to a set of M weighted samples $\{\boldsymbol{x}^{(m)}, w^{(m)}\}_{m=1}^{M}$ and obtain the

principal components $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_d\}$, indexed in descending order of variance after projection, with $\boldsymbol{p}_i \in \mathbb{R}^d$ for $\forall i$. We project the samples onto the ith principal component $\boldsymbol{p}_i$ and obtain a set of M weighted one-dimensional samples, $\{\boldsymbol{p}_i^T \boldsymbol{x}^{(m)}, w^{(m)}\}_{m=1}^M = \{x_{(i)}^{(m)}, w^{(m)}\}_{m=1}^M$.

3.4.2 Adaptive splitting

Splitting a cluster of one-dimensional samples is equivalent to finding a scalar point as the boundary. We prefer to split a cluster between modes rather than at or near a mode, because a mode of the samples potentially corresponds to the center of a component in the Gaussian mixture to be learned. Existing work chooses the mean or the median of the samples as the boundary, which is an uninformed strategy, because it is uncertain whether the mean or median falls between modes. Also, we prefer not to further split a cluster if it has only one mode, because the cluster might consist of a single component. Existing work splits a cluster without knowing its structure and checks whether the split is reasonable later in an additional tree search step, which results in extra computation involved in both overbuilding the tree and searching over the overbuilt tree. To build the tree intelligently and efficiently, we propose to learn the structure of a cluster before splitting.

To learn the structure, we apply a clustering algorithm to the projected samples. Since the tree is binary, we need only a two-component clustering algorithm. With the number of potential components specified, non-adaptive methods, such as the EM algorithm, could be used. However, most of these methods work under the assumption that the cluster should indeed be split, and thus are unable to determine whether to stop splitting. Also, most traditional methods are designed for general cases and unable to take advantage of the properties of one-dimensional data. To decide whether to stop splitting while taking

advantage of one-dimensional data, we use kernel density estimation (KDE) [47] for the embedded clustering.

We first learn the statistical distribution of the projected samples via weighted KDE as follows:

$$\hat{f}_h(x) = \sum_{m=1}^{M} w^{(m)} \phi_h(x, x_{(i)}^{(m)}), \tag{3.6}$$

where ϕ_h is a kernel function with a smoothing parameter h, and $\hat{f}_h$ is an estimate of the underlying distribution of the samples given h. Intended to learn a Gaussian mixture, we set ϕ_h as a Gaussian kernel, so that h is the standard deviation of the Gaussian kernel. h is adaptively determined according to the following approximation of the minimum mean integrated squared error (MISE) rule [47],

$$h = (\frac{4}{3M})^{1/5}\sigma, \tag{3.7}$$

where σ is the standard deviation of the underlying distribution and substituted with a robust estimate [47]:

$$\hat{\sigma} = \text{median}\left\{\left|x_{(i)}^{(m)} - \text{median}\{x_{(i)}^{(m)}, w^{(m)}\}\right|, w^{(m)}\right\}/0.6745. \tag{3.8}$$

$\hat{f}_h$ is a continuous function, and we evaluate it on a one-dimensional grid of L points. The range of the grid is set as 110% that of the samples, and given a fixed range, L determines the resolution of the discrete approximation.

Then, we look for a grid point to split the samples at. As introduced in Section 3.4.2, it is undesirable to split at or near a mode, because a mode potentially corresponds to the center of a component in the Gaussian mixture to be learned. Hence, an ideal place to split

at is a local minimum, which is a potential boundary between two adjacent components. If strict local minima are identified, we split at the one with the highest negative prominence. Otherwise, the samples could consist of one single component or multiple components close to each other. When two components stay close to each other, there could be a "plain" between them, where consecutive grid points have similar densities. A plain could result in a local minimum if tilted to the left or right. We split at the most negatively prominent one of the thus formed minima if there are any; otherwise, we consider the samples as a single component and do not split it for now.

This adaptive splitting approach applies to samples projected onto any dimension. We first consider the first principal component; if no split is performed for the first principal component, we consider the second principal component before we make a final decision.

3.4.3 Gaussian mixture model

A cluster that we do not further split corresponds to a leaf in the hierarchy tree. Every leaf corresponds to a component in the learned Gaussian mixture as follows:

$$\eta(\boldsymbol{x}) \approx \sum_{c=1}^{C} \alpha_c \, \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right), \tag{3.9}$$

where C is the total number of leaves, α_c is the sum of sample weights in the cth leaf, $\boldsymbol{\mu}_c$ is the sample mean of the cth leaf before projection, and $\boldsymbol{\Sigma}_c$ is the sample covariance matrix of the cth leaf before projection.

3.4.4 Computational complexity

In adaptive splitting, KDE costs $O(LM)$, and splitting costs $O(M)$. Hence, adaptive splitting costs $O(LM)$ overall. Non-adaptive splitting, costing $O(M)$, has a lower complexity than adaptive splitting in each recursive step of tree building, but adaptive splitting prevents overbuilding the tree and eliminates searching over the tree, thus saving considerable computation in return. Also, because adaptive splitting actively looks for Gaussian components during tree building, Gaussian mixtures learned under adaptive splitting need fewer EM iterations to reach local maximum likelihood than those learned under non-adaptive splitting, which again saves computation.

3.5 Numerical examples

We tested the proposed adaptive Gaussian mixture learning method on numerical examples of posterior-based distributed particle filtering, in comparison with other methods, to demonstrate the advantages of the proposed method in both accuracy and efficiency.

3.5.1 General settings

We considered a network of 20 sensors tracking a common moving target. The target followed a constant velocity motion model with additive Gaussian noise in a two-dimensional space.

Its state transition function was $\boldsymbol{g}(\boldsymbol{x}_n) = \boldsymbol{D} \cdot \boldsymbol{x}_n$, where

$$\boldsymbol{x}_n = \begin{bmatrix} x_{n,1} \\ x_{n,2} \\ \dot{x}_{n,1} \\ \dot{x}_{n,2} \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

and its transition noise was $\boldsymbol{u}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$, where

$$\boldsymbol{R} = 0.5 \times \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{bmatrix}. \tag{3.11}$$

The location of the kth agent, S_k, was denoted as $\boldsymbol{l}_k = (l_{k,1}, l_{k,2})$. Its observation function was assumed to be

$$\boldsymbol{h}_k(\boldsymbol{x}_n) = \sqrt{(x_{n,1} - l_{k,1})^2 + (x_{n,2} - l_{k,2})^2}, \tag{3.12}$$

and its observation noise $\boldsymbol{v}_{n,k}$ followed $\mathcal{N}(0, 0.25)$. We used 6,000 particles for local particle filtering, 15,000 samples for importance sampling in distributed fusion, 1,000 grid points for KDE, and 40 iterations for average consensus. We assumed prior knowledge of the initial state $\boldsymbol{x}_1$ to be $\mathcal{N}(\boldsymbol{g}(\boldsymbol{x}_0), \boldsymbol{R})$, with $\boldsymbol{x}_0 = [0\,\mathrm{m}, 0\,\mathrm{m}, 1\,\mathrm{m/s}, 1\,\mathrm{m/s}]^T$.

We compared the proposed method (denoted as KDE-EM) with its non-adaptive splitting counterpart (denoted as Mean-EM), hierarchical clustering methods without using the EM algorithm (denoted as KDE and Mean), and the EM algorithm with a random initial guess of 4 components (denoted as EM).

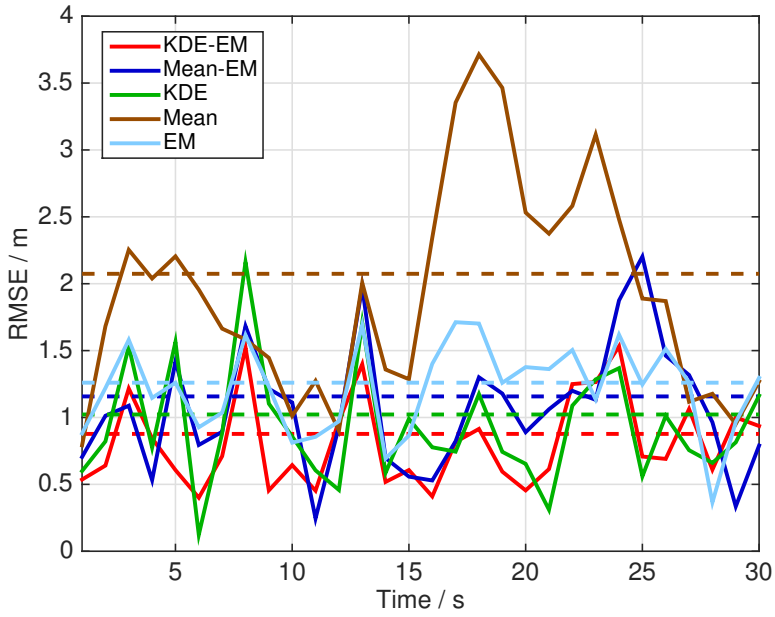


Figure 3.1: State estimation RMSE as a function of time, with dotted lines representing average RMSEs.

3.5.2 Performance

Fig. 3.1 compares the errors of distributed particle filtering using different Gaussian mixture learning methods. The proposed method (KDE-EM) yielded the lowest average root-mean-square error (RMSE) of all. More specifically, KDE-EM yielded a lower average error than Mean-EM, which demonstrates the advantage of adaptive splitting; KDE-EM yielded a lower average error than EM, which shows the advantage of informed initialization for the EM algorithm; KDE-EM and Mean-EM yielded lower average errors than KDE and Mean, respectively, which validates the benefit of adding the EM algorithm to hierarchical clustering; KDE yielded a much lower average error than Mean, which implies that the proposed adaptive splitting approach significantly improves the accuracy hierarchical clustering.

Fig. 3.2 compares the numbers of EM iterations needed in KDE-EM and Mean-EM. We can see that KDE leads to many fewer EM iterations than Mean, which verifies our previous claim that adaptive splitting helps hierarchical clustering to provide a more informed initial guess for the EM algorithm and thus to reduce the number of needed EM iterations and the amount of involved computation.

Fig. 3.3 compares the average runtimes of all the investigated methods tested under the same computing environment on the same sets of samples during distributed particle filtering. As we can see, the runtime of KDE was slightly lower than that of Mean, which verifies the saving of computation in both avoiding overbuilding the tree and eliminating searching over the tree. Also, the EM algorithm following KDE costed much more time than that following Mean, which coincides with the results in Fig. 3.2 and verifies the advantage of adaptive splitting in computational efficiency. Moreover, the EM algorithms initialized with hierarchical clustering costed less time than that initialized in a predefined way, which implies the advantage of adaptive initialization.

The communication cost of a distributed particle filtering algorithm is strongly affected by the number of components in each transmitted Gaussian mixture. In the examples above, distributed particle filtering with KDE-EM transmitted 411 Gaussian components per agent per time step, while distributed particle filtering with Mean-EM transmitted 721 components, which shows that the proposed method is more efficient in communication than its non-adaptive splitting counterpart.

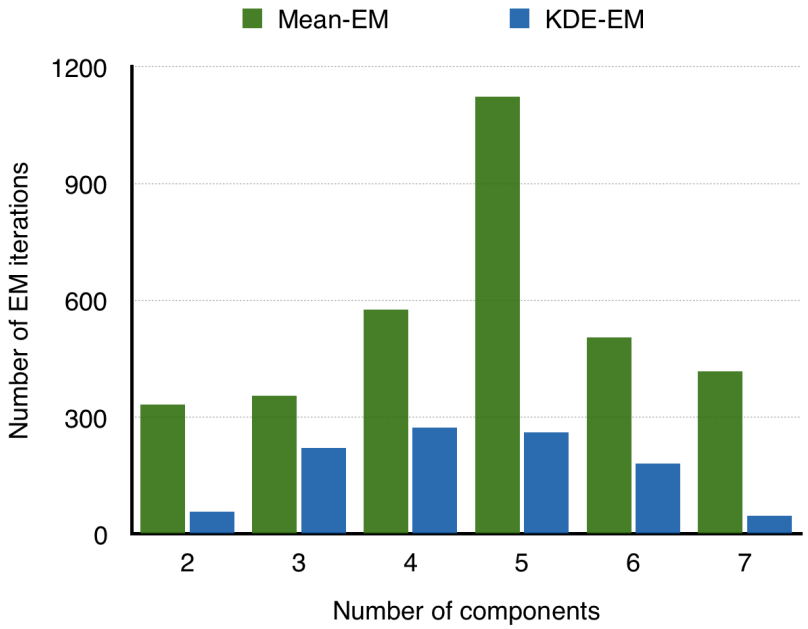


Figure 3.2: The number of EM iterations as a function of the number of components in a Gaussian mixture.

3.6 Chapter summary

In this chapter, we combined hierarchical clustering with the EM algorithm for adaptive Gaussian mixture learning in posterior-based distributed particle filtering. We designed a one-dimensional clustering algorithm based on kernel density estimation and embedded the designed clustering algorithm in each recursive step of the hierarchical clustering to adaptively determine whether and where to split a cluster. This adaptive splitting approach improves the accuracy of hierarchical clustering and saves computation in both hierarchical clustering and the following EM algorithm. We tested the proposed adaptive Gaussian mixture learning method, in comparison with other methods, on numerical examples of distributed particle filtering. The numerical examples validated the advantage of adaptive Gaussian mixture learning over non-adaptive Gaussian mixture learning, demonstrated the

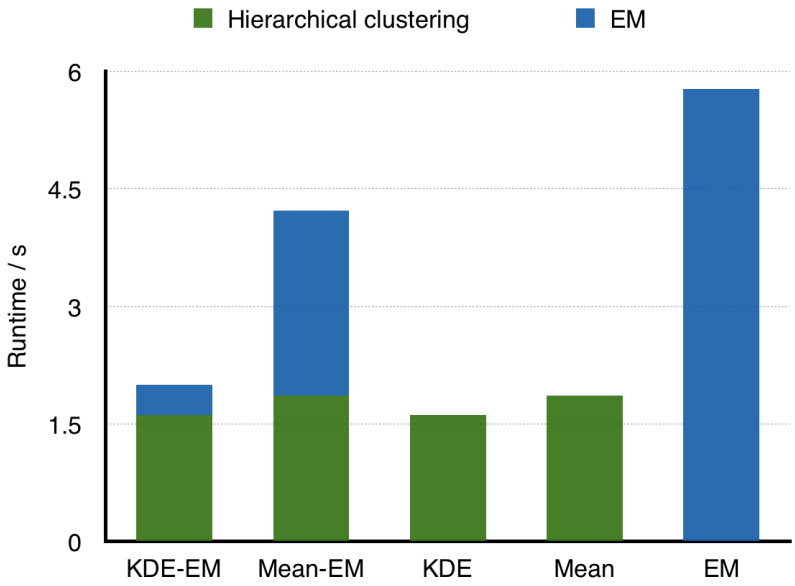


Figure 3.3: The average runtime of each method tested on the same sets of samples under the same computing environment.

benefits of adaptive splitting in hierarchical clustering, verified the importance of the EM algorithm for adaptive Gaussian mixture learning, and showed the advantages of the proposed method in estimation accuracy, computational efficiency, and communication efficiency. In the numerical examples, the runtime of the EM algorithm initialized with adaptive splitting was 7% that of the EM algorithm initialized randomly and 17% that of the EM algorithm initialized with non-adaptive splitting, which implies that adaptive splitting results in a highly informed initial guess for the EM algorithm and significantly improves the computational efficiency of Gaussian mixture learning.

In addition to distributed particle filtering, the proposed adaptive Gaussian mixture learning method can be used in any other applications where a Gaussian mixture representation is needed. Also, the proposed method can be used to solve clustering problems where the number of clusters is unknown.

Chapter 4

Clock Synchronization in Wireless Sensor Networks

We propose a method to jointly estimate sequential target states and the network synchronization status based on observations obtained by an unsynchronized wireless sensor network. We build an unsynchronized multi-sensor state-space model to connect asynchronous sensor observations with target state transition. Under the built model, we solve the joint estimation problem via the expectation-maximum (EM) algorithm, assuming a known temporal order of sensor clocks. Based on the solution and a hypothesis testing method developed for temporal ordering, we solve the joint estimation problem in a distributed manner, assuming an unknown temporal order. We use Monte Carlo methods to approximate our solutions, in order to deal with nonlinear models and non-Gaussian noise. Moreover, we develop a recursive and parallel algorithm to compute the EM covariance matrix under Monte Carlo approximations. Numerical examples demonstrate the performance of the proposed method

and show that sequential target estimation benefits from considering clock synchronization. [3]

4.1 Introduction

Wireless sensor networks have been widely used in sequential target estimation [48]-[51]. Collaboration among networked sensors provides observations from multiple perspectives, thus enhancing estimation accuracy. However, effective collaboration strongly relies on perfect synchronization between sensors. Synchronization is difficult to maintain, because the internal clock of a sensor is very likely to drift away from its initial setting as time goes on. Simply ignoring the fact that a sensor network is unsynchronized will lead to estimation errors. Therefore, accurate estimation of the synchronization status of a sensor network is essential for reliable sequential estimation of target states. Such joint estimation of target states and synchronization status can be achieved using statistical signal processing methods.

A possible solution to the joint estimation problem is to first learn the synchronization status using a clock synchronization method [22], [23] and then estimate the target trajectory based on sensor observations together with the learned synchronization status [52]. However, this solution has three major drawbacks. First, with no additional information given, the clock synchronization method has to be based on repeated communication of timestamps [53], [54], which expends considerable energy and is thus undesirable for a wireless sensor network, which has a limited energy budget. Second, since the target to be observed often appears unexpectedly in real-world (e.g., military) applications, the synchronization process

[3] This chapter is based on J. Li and A. Nehorai, "Joint sequential target estimation and clock synchronization in wireless sensor networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 74–88, June 2015.

and the observation process are always separated in time; in other words, there is always a time gap of uncertain length between synchronization and observation. If the gap is not sufficiently small, clocks can drift during the gap, the learned synchronization status can become outdated, and thus the sequential target estimation will be inaccurate. Third, the target estimation step is not robust to errors or failures in the clock synchronization step.

To overcome the drawbacks, we propose to jointly estimate both target states and synchronization status based on asynchronous sensor observations. To the best of the authors' knowledge, there is no existing work on observation-based clock synchronization. But observation-based clock synchronization is totally realizable, because sensor observations are taken according to the sensors' own clocks, and thus convey implicit temporal information about relative offsets between clocks. The proposed joint estimation approach has three major advantages over the above-mentioned synchronize-then-observe approach. First, sensor observations are available in any sequential target estimation task, so there is no need for further information communication or data collection, which saves a substantial amount of energy. Second, observation-based clock synchronization guarantees that the joint estimation approach learns the exact synchronization status under which sensor observations are taken. Third, the joint estimation approach works effectively with unknown synchronization status, and is thus robust to errors or failures in collecting prior knowledge. These facts motivate us to follow such an observation-based joint estimation approach.

A joint estimation problem is often solved with the augmented state-space method [55], which appends the parameter vector to the state vector and incorporates an artificial parameter evolution function in the state transition. This method simplifies the original problem into a single state estimation problem, but adds to the dimensions of the state vector. In this chapter, we use an alternating optimization approach to split the original problem into separate

state and parameter estimation problems that alternate until convergence. In particular, we use the expectation-maximization (EM) algorithm [27] as a tool for alternating optimization. We formulate the joint estimation problem by building an unsynchronized multi-sensor state-space model to connect asynchronous sensor observations with corresponding state transition. Based on the model, we derive both a centralized solution to the joint estimation problem with a known temporal order and a distributed solution to the joint estimation problem with an unknown temporal order. The distributed solution determines local temporal orders via hypothesis testing, then uses the centralized solution to estimate local synchronization statuses based on local sensor observations, and finally fuses local estimates to form a global estimate of the network synchronization status, based on which a final estimate of the target trajectory is obtained. The distributed nature of the method makes it scalable.

The EM algorithm, which we use for joint estimation, has long been criticized for its difficulty in computing the EM covariance matrix. However, knowing the EM covariance matrix enables us to evaluate statistical performance, to estimate the fraction of missing information involved in a problem, to compute the asymptotic global rate of convergence, and to speed up convergence [56]. In [57], Louis proposes a primal method to compute the EM covariance matrix via the inverse of the Fisher information, but Louis's formula is computationally intractable. Later researchers proposed methods to circumvent calculating Louis's formula [58]-[60], but they are subject to numerical inaccuracies and instabilities. In this chapter, we propose an algorithm to compute the intractable term in Louis's formula in a recursive and parallel fashion under Monte Carlo approximations.

This chapter makes three major contributions. First, we propose a novel approach to clock synchronization. We estimate clock synchronization status based on sequential sensor observations, rather than the traditionally used timestamps obtained from repeated message

passing. This method significantly reduces inter-sensor communication, saves energy and resources, and works perfectly with sequential target estimation. Second, we design an unsynchronized multi-sensor state-space model to study the interaction between state transition and asynchronous observations, and we solve a joint estimation problem under the designed model, using the EM algorithm and the Monte Carlo method. Third, we propose an algorithm to compute the EM covariance matrix based on a primal method that was previously considered to be intractable. The proposed algorithm is designed under our problem formulation, but is also extendable to other parameter estimation problems under state-space models.

The rest of the chapter is organized as follows. Section 4.2 introduces the unsynchronized multi-sensor state-space model. Section 4.3 derives an EM solution to the joint estimation problem with a known temporal order. Section 4.4 approximates the EM solution using Monte Carlo methods and discusses stochastic variants of the EM algorithm. Section 4.5 derives a distributed solution to the joint estimation problem with an unknown temporal order. Section 4.6 proposes an algorithm to compute the EM covariance matrix. Section 4.7 presents numerical examples, and Section 4.8 concludes the chapter.

4.2 Signal models

4.2.1 Clock model

We model a clock C as an affine function of time t,

$$C(t) = \lambda \cdot t + \tau, \tag{4.1}$$

where λ is the clock skew and τ is the clock offset. Model (4.1) enables us to model a clock C_1 as an affine function of another clock C_2 through

$$C_1(t) = \lambda_{1,2} \cdot C_2(t) + \tau_{1,2}, \tag{4.2}$$

where $\lambda_{1,2}$ is the relative clock skew and $\tau_{1,2}$ is the relative clock offset. C_1 and C_2 are synchronized if $\lambda_{1,2} = 1$ and $\tau_{1,2} = 0$. To achieve synchronization between clocks, the relative clock skew $\lambda_{1,2}$ and the relative clock offset $\tau_{1,2}$ have to be estimated. Since a relative clock skew is determined by the frequencies of internal crystal oscillators and is thus relatively robust to interference in the long term, we assume it to be 1 for each pair of clocks to be investigated in this chapter. Also, we assume a relative clock offset to be constant during the investigated period. Under these assumptions, unsynchronized sensors programmed to collaborate in simultaneous and periodic sampling will take observations at the same intervals but with fixed offsets in time.

4.2.2 Synchronized multi-sensor state-space model

Denote the sensors under investigation as $S_1, S_2, \ldots, S_K$, where K is the total number of sensors, and the relative clock offset between the S_i and S_j as $\tau_{i,j}$. By definition, we have $\tau_{i,j} = -\tau_{j,i}$ and $\tau_{i,i} = 0$. A sensor network is synchronized if $\tau_{i,j} = 0$ for $\forall i, j$. In a synchronized network, sensors take observations simultaneously, and always observe the same state of a common target. In this case, a multi-sensor state-space model can be built as

$$\begin{cases} \boldsymbol{x}_t = \boldsymbol{g}(\boldsymbol{x}_{t-\Delta t}, \Delta t) + \boldsymbol{u}_t(\Delta t) \\ \boldsymbol{y}_{t,k} = \boldsymbol{h}_k(\boldsymbol{x}_t) + \boldsymbol{v}_{t,k} \quad (k = 1, 2, \ldots, K) \end{cases}, \tag{4.3}$$

where

1) $\boldsymbol{x}_t \in \mathbb{R}^d$ is the target state at time t;

2) $\boldsymbol{y}_{t,k} \in \mathbb{R}^b$ is the observation taken by S_k at time t;

3) Δt is the state transition interval or the observation period;

4) $\boldsymbol{g}$ is a known twice differentiable state transition function;

5) $\boldsymbol{h}_k$ is a known observation function of S_k;

6) $\{\boldsymbol{u}_t\}$ and $\{\boldsymbol{v}_{t,k}\}$ are uncorrelated additive noise;

8) $\{\boldsymbol{u}_t\}$ and $\{\boldsymbol{v}_{t,k}\}$ are independent and belong to the exponential family with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}_k$, i.e., the pdf of $\boldsymbol{u}_t$ is $f(\boldsymbol{u}_t|\boldsymbol{\theta}) = c_u(\boldsymbol{\theta})a_u(\boldsymbol{u}_t)\exp\left(\boldsymbol{\pi}_u(\boldsymbol{\theta})^T\boldsymbol{\lambda}_u(\boldsymbol{u}_t)\right)$, and that of $\boldsymbol{v}_{t,k}$ is $f(\boldsymbol{v}_{t,k}|\boldsymbol{\phi}_k) = c_v(\boldsymbol{\phi}_k)a_v(\boldsymbol{v}_{t,k})\exp\left(\boldsymbol{\pi}_v(\boldsymbol{\phi}_k)^T\boldsymbol{\lambda}_v(\boldsymbol{v}_{t,k})\right)$;

9) $\boldsymbol{\theta}$ is further parametrized by Δt with $\boldsymbol{\theta}(\Delta t)$ being twice differentiable, while $\boldsymbol{\phi}_k$ is a constant parameter;

10) state transition is Markovian, i.e., $\boldsymbol{x}_t \perp\!\!\!\perp \boldsymbol{x}_{t'} \mid \boldsymbol{x}_{t''}$ for $t < t'' < t'$ ("$\perp\!\!\!\perp$" means "be independent of," and "|" means "given").

4.2.3 Unsynchronized multi-sensor state-space model

A sensor network is unsynchronized, if $\tau_{i,j} \neq 0$ for $\forall i, j$ with $i \neq j$. Since clock offsets are normally much smaller than the pre-assumed observation period (otherwise they would be easily detected), we assume that $|\tau_{i,j}| < \Delta t$ for $\forall i, j$. Also, we assume for now that we know the temporal order of the sensors, so that we could model state transition sequentially in time. More specifically, we re-index the sensors according to their temporal order, so that a sensor with a smaller index is ahead in time, i.e., $\tau_{k,k+1} > 0$ or $C_k(t) > C_{k+1}(t)$ for $\forall k$. In this case, we can sequentially model the interaction between state transition and sensor observations as

$$\begin{cases} \boldsymbol{x}_{t,k} = \boldsymbol{g}(\boldsymbol{x}_{t,k-1}, \tau_{k-1,k}) + \boldsymbol{u}_{t,k}(\tau_{k-1,k}) \\ \boldsymbol{x}_{t+\Delta t,1} = \boldsymbol{g}(\boldsymbol{x}_{t,K}, \Delta t - \tau_{1,K}) + \boldsymbol{u}_{t+\Delta t,1}(\Delta t - \tau_{1,K}) \\ \boldsymbol{y}_{t,k} = \boldsymbol{h}_k(\boldsymbol{x}_{t,k}) + \boldsymbol{v}_{t,k} \end{cases}, \tag{4.4}$$

where

1) $\boldsymbol{x}_{t,k}$ is the target state at time $t + \tau_{1,k}$,

2) $\boldsymbol{y}_{t,k}$ is the observation of $\boldsymbol{x}_{t,k}$ or the observation taken by S_k during the period starting from time t,

3) $\boldsymbol{u}_{t,k}$ is the noise to the transition from the last state to $\boldsymbol{x}_{t,k}$,

4) $\boldsymbol{v}_{t,k}$ is the observation noise of S_k when taking $\boldsymbol{y}_{t,k}$, and

5) all the conditional independence properties of (4.3) still hold.

We simplify the notation $\tau_{k-1,k}$ to τ_{k-1} and $\Delta t - \tau_{1,K}$ to τ_K, implying that τ_k ($k \in \{1, 2, \ldots, K\}$) represents the time interval between an observation by S_k and the next observation taken by the network. Also, since all relative clock offsets are assumed to be smaller than the observation period Δt, we normalize all the time variables in the model by Δt; since the starting time of each observation period is a multiple of Δt, we discretize all the time indices to integers. After simplification, normalization, and discretization, model (4.4) becomes

$$\begin{cases} \boldsymbol{x}_{n,k} = \boldsymbol{g}(\boldsymbol{x}_{n,k-1}, \tau_{k-1}) + \boldsymbol{u}_{n,k}(\tau_{k-1}) \\ \boldsymbol{x}_{n+1,1} = \boldsymbol{g}(\boldsymbol{x}_{n,K}, \tau_K) + \boldsymbol{u}_{n+1,1}(\tau_K) \\ \boldsymbol{y}_{n,k} = \boldsymbol{h}_k(\boldsymbol{x}_{n,k}) + \boldsymbol{v}_{n,k} \end{cases}, \tag{4.5}$$

where

1) n is the observation period index, i.e., $n = \left\lfloor \frac{t}{\Delta t} \right\rfloor + 1$;

2) $n = 1, 2, \ldots, N$, where N is the total number of observation periods;

3) $\boldsymbol{x}_{n,k}$ is the target state observed by S_k during the nth observation period;

4) $\boldsymbol{y}_{n,k}$ is the observation by S_k in the nth observation period;

5) $\sum_{i=1}^{K} \tau_i = 1$.

4.3 Joint estimation with a known temporal order

In this section, we solve the joint estimation problem, assuming that we know the temporal order of the sensors. Our goal is to estimate the target states $\{\boldsymbol{x}_{n,k}\}$ and the relative clock offsets $\{\tau_{i,j}\}$, given the observations $\{\boldsymbol{y}_{n,k}\}$. Although the relative clock offsets explicitly determine the varying state transition intervals and thus play a more important role than ordinary parameters in a dynamic model, we treat them as unknown parameters and cast the problem into a joint state and parameter estimation problem. We solve this joint estimation problem offline with an alternating estimation strategy. In particular, we use the EM algorithm.

We use the following notations in the rest of the chapter:

$$\begin{aligned}
\boldsymbol{\tau} &= [\tau_1, \tau_2, \ldots, \tau_{K-1}]^T, \\
\boldsymbol{X} &= \{\boldsymbol{x}_{n,k} : 1 \le n \le N,\ 1 \le k \le K\}, \text{and} \\
\boldsymbol{Y} &= \{\boldsymbol{y}_{n,k} : 1 \le n \le N,\ 1 \le k \le K\}.
\end{aligned}$$

Note that $\tau_1, \tau_2, \ldots, \tau_{K-1}$ are all the offsets we need to estimate, since any other offset can be calculated from them, e.g., $\tau_K = 1 - \sum_{k=1}^{K-1} \tau_k$ and $\tau_{i,j} = \sum_{k=i}^{j-1} \tau_k$ for $i < j$.

4.3.1 Expectation-maximization algorithm

The EM algorithm is an iterative parameter estimation method for maximum likelihood estimation (MLE) with incomplete or missing data. The MLE solution to our problem under the setting of parameter estimation can be expressed as

$$\hat{\boldsymbol{\tau}} = \arg\max_{\boldsymbol{\tau}} L(\boldsymbol{\tau}; \boldsymbol{Y}). \tag{4.6}$$

With the intermediate variable $\boldsymbol{X}$ missing, it is usually intractable to compute the observed likelihood $L(\boldsymbol{\tau}; \boldsymbol{Y})$. The EM algorithm is used to solve this problem by circumventing the intractability.

Each iteration of the EM algorithm consists of an expectation step (E-step) and a maximization step (M-step). In the E-step, given $\boldsymbol{Y}$ and the estimate of $\boldsymbol{\tau}$ in the last iteration, the conditional distribution of $\boldsymbol{X}$ is computed, and the expectation of the complete log-likelihood $\log L(\boldsymbol{\tau}; \boldsymbol{X}, \boldsymbol{Y})$ is taken with respect to $\boldsymbol{X}$ over its conditional distribution, so that the unknown auxiliary variable $\boldsymbol{X}$ vanishes and the observed log-likelihood $\log L(\boldsymbol{\tau}; \boldsymbol{Y})$ is approximated. In the M-step, the observed likelihood approximated via expectation in the E-step is maximized with respect to $\boldsymbol{\tau}$, and the current estimate of $\boldsymbol{\tau}$ is replaced with the maximizer. These two steps repeat until the sequence of parameter estimates converges.

In our problem, the complete data is $(\boldsymbol{X}, \boldsymbol{Y})$, the missing data is $\boldsymbol{X}$, and the parameter to be iteratively estimated is $\boldsymbol{\tau}$. Thus, the EM algorithm can be formulated as

$$\text{E-Step} : Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) = \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\hat{\boldsymbol{\tau}}^{(i)}} \log L(\boldsymbol{\tau}; \boldsymbol{X}, \boldsymbol{Y}) \tag{4.7}$$

$$\text{M-Step} : \hat{\boldsymbol{\tau}}^{(i+1)} = \arg\max_{\boldsymbol{\tau}} Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}), \tag{4.8}$$

where $\hat{\boldsymbol{\tau}}^{(i)}$ is the estimate of $\boldsymbol{\tau}$ from the ith iteration, and $Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$ is the objective function of the $(i+1)$th iteration. The algorithm starts with an initial guess $\hat{\boldsymbol{\tau}}^{(0)}$ and ends when the sequence $\{\hat{\boldsymbol{\tau}}^{(i)}\}$ converges.

Note that the posterior distribution of $\boldsymbol{X}$ is computed in the E-step, based on the latest estimate of $\boldsymbol{\tau}$. In other words, the state variable $\boldsymbol{X}$ is estimated in each iteration of the EM algorithm, thus making the EM algorithm suitable for joint state and parameter estimation, although it is originally designed for parameter estimation.

4.3.2 Objective function

The objective function of an iteration is the conditional expectation of the complete log-likelihood. Using the joint distribution factorization property of a Bayesian network [61], the complete likelihood can be computed as

$$L(\boldsymbol{\tau}; \boldsymbol{X}, \boldsymbol{Y}) = \prod_{n=1}^{N} \prod_{k=1}^{K} f(\boldsymbol{y}_{n,k} | \boldsymbol{pa}(\boldsymbol{y}_{n,k}); \boldsymbol{\tau}) f(\boldsymbol{x}_{n,k} | \boldsymbol{pa}(\boldsymbol{x}_{n,k}); \boldsymbol{\tau}), \tag{4.9}$$

where f is the notation for probability density functions (pdf), and the function $\boldsymbol{pa}$ outputs the set of parents of the input variable in the Bayesian network. In a Bayesian network, a parent of a variable is another variable on which it directly depends. In our model, $\boldsymbol{pa}(\boldsymbol{y}_{n,k}) = \{\boldsymbol{x}_{n,k}\}$; $\boldsymbol{pa}(\boldsymbol{x}_{n,k}) = \{\boldsymbol{x}_{n,k-1}\}$ for $k = 2, 3, \ldots, K$ and $n = 1, 2, \ldots, N$; $\boldsymbol{pa}(\boldsymbol{x}_{n+1,1}) = \{\boldsymbol{x}_{n,K}\}$ for $n = 1, 2, \ldots, N-1$; and $\boldsymbol{pa}(\boldsymbol{x}_{1,1}) = \varnothing$. Thus, the complete likelihood can be rewritten

as

$$L(\boldsymbol{\tau}; \boldsymbol{X}, \boldsymbol{Y}) = \prod_{n=1}^{N} \prod_{k=1}^{K} f(\boldsymbol{y}_{n,k} | \boldsymbol{x}_{n,k}) \times \prod_{n=1}^{N-1} f(\boldsymbol{x}_{n+1,1} | \boldsymbol{x}_{n,K}; \boldsymbol{\tau}) \times \prod_{n=1}^{N} \prod_{k=1}^{K-1} f(\boldsymbol{x}_{n,k+1} | \boldsymbol{x}_{n,k}; \boldsymbol{\tau}) \times f(\boldsymbol{x}_{1,1}). \tag{4.10}$$

Then, the objective function of the $(i+1)$th iteration can be calculated as

$$\begin{aligned} Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) &= \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)}} \log L(\boldsymbol{\tau}; \boldsymbol{X}, \boldsymbol{Y}) \\ &= \underbrace{\sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E} \log f(\boldsymbol{y}_{n,k} | \boldsymbol{x}_{n,k})}_{\text{first term}} + \underbrace{\sum_{n=1}^{N-1} \mathbb{E} \log f(\boldsymbol{x}_{n+1,1} | \boldsymbol{x}_{n,K}; \boldsymbol{\tau})}_{\text{second term}} \\ &\quad + \underbrace{\sum_{n=1}^{N} \sum_{k=1}^{K-1} \mathbb{E} \log f(\boldsymbol{x}_{n,k+1} | \boldsymbol{x}_{n,k}; \boldsymbol{\tau})}_{\text{third term}} + \underbrace{\mathbb{E} \log f(\boldsymbol{x}_{1,1})}_{\text{fourth term}}, \end{aligned} \tag{4.11}$$

where the subscript "$\boldsymbol{X}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)}$" is omitted from each expectation symbol in (4.11) and later expressions for simplicity of notation.

Note that there are four terms in the objective function (4.11): the first term is for observation, the second and third terms are for state transition, and the fourth term is for prior information. Since only state transition directly depends on $\boldsymbol{\tau}$, maximizing the objective function $Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$ with respect to $\boldsymbol{\tau}$ is equivalent to maximizing the sum of the second and third terms in the objective functions. In other words, let

$$\begin{aligned} Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) &= \sum_{n=1}^{N-1} \mathbb{E} \log f(\boldsymbol{x}_{n+1,1} | \boldsymbol{x}_{n,K}; \boldsymbol{\tau}) \\ &\quad + \sum_{n=1}^{N} \sum_{k=1}^{K-1} \mathbb{E} \log f(\boldsymbol{x}_{n,k+1} | \boldsymbol{x}_{n,k}; \boldsymbol{\tau}), \end{aligned} \tag{4.12}$$

then we have $\arg\max_{\boldsymbol{\tau}} Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) = \arg\max_{\boldsymbol{\tau}} Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$, and the M-step is equivalent to

$$\hat{\boldsymbol{\tau}}^{(i+1)} = \arg\max_{\boldsymbol{\tau}} Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}). \tag{4.13}$$

We call $Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$ the partial objective function.

Since the distribution of state transition noise is known, the only thing unknown in (4.12) is the posterior distribution of $\boldsymbol{X}$ with which the expectations are computed, i.e., $f(\boldsymbol{X}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$. It further simplifies to $f(\boldsymbol{x}_{n,k+1}, \boldsymbol{x}_{n,k}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$ and $f(\boldsymbol{x}_{n+1,1}, \boldsymbol{x}_{n,K}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$, because state transition relates only two adjacent states at a time. Our next goal is to compute the joint probability density functions $f(\boldsymbol{x}_{n,k+1}, \boldsymbol{x}_{n,k}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$ and $f(\boldsymbol{x}_{n+1,1}, \boldsymbol{x}_{n,K}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$.

4.3.3 Smoothing

An efficient approach to calculating a joint conditional density is to calculate its marginal conditional densities and use the chain rule combined with conditional independence properties to derive the joint conditional density. Thus, our next step is to calculate $f(\boldsymbol{x}_{k,n}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$ for all k and n, which is often recognized as a smoothing problem.

For simplicity, we introduce two alternative rules of notation. First, we re-index the state and observation variables using one-dimensional subscripts in the order of time, i.e., denoting $\boldsymbol{x}_{n,k}$ and $\boldsymbol{y}_{n,k}$ as $\boldsymbol{x}_{K(n-1)+k}$ and $\boldsymbol{y}_{K(n-1)+k}$, respectively, for all k and n. Second, we denote the set of consecutive observations from $\boldsymbol{y}_i$ to $\boldsymbol{y}_j$ $(i < j)$ as $\boldsymbol{y}_{i:j}$. In this way, the marginal conditional densities to be computed can be expressed as $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:KN}; \hat{\boldsymbol{\tau}}^{(i)})$ for $n = 1, 2, \ldots, KN$.

We use the forward-backward algorithm [62] to solve the smoothing problem iteratively. The forward part of the algorithm obtains a filtering result $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}; \hat{\boldsymbol{\tau}}^{(i)})$ based on the

preceding one $f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)})$ for $n = 2, 3, \ldots, KN$; the backward part of the algorithm obtains a smoothing result $f(\boldsymbol{x}_n|\boldsymbol{Y};\hat{\boldsymbol{\tau}}^{(i)})$ based on the succeeding one $f(\boldsymbol{x}_{n+1}|\boldsymbol{Y};\hat{\boldsymbol{\tau}}^{(i)})$ for $n = KN-1, KN-2, \ldots, 1$.

The forward part proceeds iteratively as follows:

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)}) \propto f(\boldsymbol{y}_n|\boldsymbol{x}_n)\int f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1};\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n-1}. \tag{4.14}$$

The backward part proceeds iteratively as follows:

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)}) = \int f(\boldsymbol{x}_n, \boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1} \tag{4.15}$$

$$= \int \frac{f(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n;\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})}{\int f(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n;\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_n} \times f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1}. \tag{4.16}$$

The derivation of (4.14) and (4.16) can be found in the Appendix.

Note that the integrand in (4.15) is exactly the joint conditional density of adjacent states and can be computed according to the integrand of (4.16). In other words, the joint conditional density comes as a byproduct of the forward-backward algorithm, which is therefore convenient and useful for our problem.

4.3.4 Maximization

The M-step solves a constrained optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\tau}} \quad & Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) \\ \text{s. t.} \quad & 0 < \tau_k < 1 \text{ for } \forall k \\ & \sum_{k=1}^{K-1} \tau_k < 1. \end{aligned}$$

The problem can be reformulated as

$$\begin{aligned} \max_{\boldsymbol{\tau}} \quad & Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) \\ \text{s. t.} \quad & \begin{bmatrix} \mathbf{1}_{K-1}^T \\ -\boldsymbol{I}_{K-1} \end{bmatrix} \boldsymbol{\tau} \prec \begin{bmatrix} 1 \\ \mathbf{0}_{K-1} \end{bmatrix}, \end{aligned} \tag{4.17}$$

where $\mathbf{1}_{K-1}$ is an all-ones column vector of length $K-1$, $\mathbf{0}_{K-1}$ is an all-zeros column vector of length $K-1$, $\boldsymbol{I}_{K-1}$ is an identity matrix of size $K-1$, and "$\prec$" means "element-wise less than".

According to the assumptions, Q_{p} is both once and twice differentiable, and it is convenient to analytically obtain $\frac{\partial}{\partial \boldsymbol{\tau}} Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$ and $\frac{\partial^2}{\partial \boldsymbol{\tau} \partial \boldsymbol{\tau}^T} Q_{\mathrm{p}}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)})$. Hence, the optimization problem in (4.17) is compatible with gradient methods. Given the linear constraints in (4.17), we use the gradient projection method [63], a gradient descent method for optimization with linear constraints, to solve the optimization problem.

4.4 Monte Carlo approximations

Although the forward-backward algorithm is easy to follow, the involved integrals can be intractable to compute, since state transition and observation functions can be nonlinear and noise can be non-Gaussian. Thus, to simplify the calculation, we use the Monte Carlo method to approximate the densities and transform the integrals into sums.

4.4.1 Particle filtering and smoothing

The Monte Carlo version of filtering is called particle filtering [36]. In particle filtering, the posterior distribution of a state is approximated by a sufficiently large number of weighted samples or, in other words, particles:

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)}) \approx \sum_{m=1}^{M_{i+1}} w_n^{(m)}\delta(\boldsymbol{x}_n - \boldsymbol{x}_n^{(m)}), \tag{4.18}$$

where

1) M_{i+1} is the number of particles used for the approximation of a single state in the $(i+1)$th iteration, and can vary with i;

2) $\boldsymbol{x}_n^{(m)}$ is the mth particle used for the approximation of $\boldsymbol{x}_n$, and is generated according to a proposal distribution [36];

3) $w_n^{(m)}$ is the weight of $\boldsymbol{x}_n^{(m)}$ with $\sum_{m=1}^{M_{i+1}} w_n^{(m)} = 1$; and

4) δ is the Dirac delta function.

Particle filtering is a Monte Carlo approximation to a simplified forward part of the forward-backward algorithm. The way in which particle weights are computed depends on the choice of a proposal distribution. Usually, the proposal distribution of $\boldsymbol{x}_n^{(m)}$ is chosen as the state

transition distribution $f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}^{(m)};\hat{\boldsymbol{\tau}}^{(i)})$, and, correspondingly, the particle weights are iteratively computed as

$$w_n^{(m)} \propto f(\boldsymbol{y}_n|\boldsymbol{x}_n^{(m)}) \cdot w_{n-1}^{(m)}, \tag{4.19}$$

where the recursion is initialized with uniform weights. Resampling of particles with replacement is performed when most of the weight is on a small group of particles.

The Monte Carlo version of smoothing is called particle smoothing [64]. Filtering and smoothing share particles, but assign different weights to them, because they are conditioned on different information. More specifically, smoothing densities are conditioned on future observations in addition to up-to-date observations that filtering densities are conditioned on. In particle smoothing, the posterior distribution of a state is approximated as

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)}) \approx \sum_{m=1}^{M_{i+1}} w_n^{(m)*}\delta(\boldsymbol{x}_n - \boldsymbol{x}_n^{(m)}), \tag{4.20}$$

where the asterisk indicates that the weight is updated with additional evidence, i.e., $\boldsymbol{y}_{(n+1):KN}$ in this case. With this in mind, the backward part in (4.16) can be rewritten as

$$\begin{aligned} w_n^{(m)*} &= f(\boldsymbol{x}_n^{(m)}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)}) \\ &= \sum_{k=1}^{M_{i+1}} \frac{f(\boldsymbol{x}_{n+1}^{(k)}|\hat{\boldsymbol{x}_n}^{(m)};\hat{\boldsymbol{\tau}}^{(i)})w_n^{(m)}}{\sum_{j=1}^{M_{i+1}} f(\boldsymbol{x}_{n+1}^{(k)}|\boldsymbol{x}_n^{(j)};\hat{\boldsymbol{\tau}}^{(i)})w_n^{(j)}} w_{n+1}^{(k)*}, \end{aligned} \tag{4.21}$$

where $w_{KN}^{(m)*} = w_{KN}^{(m)}$ for all m upon initialization.

Similarly, the joint conditional probability density function can be approximated as

$$f(\boldsymbol{x}_n, \boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN}; \hat{\boldsymbol{\tau}}^{(i)}) \approx \sum_{k=1}^{M_{i+1}} \sum_{j=1}^{M_{i+1}} w_{n,n+1}^{(j,k)*} \delta(\boldsymbol{x}_n - \boldsymbol{x}_n^{(j)}) \delta(\boldsymbol{x}_{n+1} - \boldsymbol{x}_{n+1}^{(k)}), \tag{4.22}$$

where according to the integrand of (4.16)

$$\begin{aligned} w_{n,n+1}^{(j,k)*} &= f(\boldsymbol{x}_n^{(j)}, \boldsymbol{x}_{n+1}^{(k)}|\boldsymbol{y}_{1:KN}; \hat{\boldsymbol{\tau}}^{(i)}) \\ &= \frac{f(\boldsymbol{x}_{n+1}^{(k)}|\boldsymbol{x}_n^{(j)}; \hat{\boldsymbol{\tau}}^{(i)}) w_n^{(j)}}{\sum_{l=1}^{M_{i+1}} f(\boldsymbol{x}_{n+1}^{(k)}|\boldsymbol{x}_n^{(l)}; \hat{\boldsymbol{\tau}}^{(i)}) w_n^{(l)}} w_{n+1}^{(k)*}. \end{aligned} \tag{4.23}$$

With $\{w_{n,n+1}^{(j,k)*}\}_{j,k=1}^{M_{i+1}}$ $(n = 1, 2, \ldots, KN-1)$ known, the conditional expectations in (4.12) can be directly calculated.

4.4.2 Stochastic variants of the EM algorithm

With densities in the EM objective function approximated by weighted random samples, the EM algorithm becomes stochastic, and hence not all of the properties of the EM algorithm still hold.

Two major stochastic variants of the EM algorithm are the Monte Carlo EM algorithm (MCEM) [65] and the stochastic approximation EM algorithm (SAEM) [66].

Monte Carlo EM algorithm

MCEM generates Monte Carlo samples $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(M_{i+1})}$ from the posterior distribution $f(\boldsymbol{X}|\boldsymbol{Y}; \hat{\boldsymbol{\tau}}^{(i)})$ and approximates the objective function as

$$Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) = \frac{1}{M_{i+1}} \sum_{m=1}^{M_{i+1}} \log L(\boldsymbol{\tau}; \boldsymbol{X}^{(m)}, \boldsymbol{Y}). \tag{4.24}$$

Stochastic approximation EM algorithm

SAEM uses the same approach to approximate Q, but considers $\bar{Q}$, a moving average of Q with forgetting factors $\{\gamma_i\}$, as its objective function:

$$\bar{Q}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}) = (1 - \gamma_i)\bar{Q}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i-1)}) + \gamma_i Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(i)}). \tag{4.25}$$

The stochastic approximation is initialized with $\bar{Q}(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(0)}) = Q(\boldsymbol{\tau}; \hat{\boldsymbol{\tau}}^{(0)})$ and continues according to (4.25), under the constraints that $0 < \gamma_i \leq 1$, $\sum \gamma_i = \infty$ and $\sum \gamma_i^2 < \infty$.

MCEM has a rate of convergence comparable to that of the EM algorithm, but is not guaranteed to converge, unless the sample size (or the number of particles) keeps increasing across iterations [67], [68], which, however, results in a serious computational concern. In contrast, SAEM has a lower rate of convergence than MCEM, but is guaranteed to converge with a finite sample size, both due to the involved moving average.

To combine the advantages of MCEM and SAEM, we first use MCEM for a relatively high rate of convergence and then use SAEM for guaranteed convergence. Since MCEM is a special case of SAEM with $\gamma_i = 1$ for $\forall i$, the combination approach is equivalent to SAEM with $\gamma_i = 1$ first and $\gamma_i < 1$ afterwards. We switch from $\gamma_i = 1$ to $\gamma_i < 1$ when the

sequence $\{\hat{\boldsymbol{\tau}}^{(j)}\}_{j=1}^{i}$ enters a neighborhood of convergence, which is quantitatively determined as follows.

We use the average variance of recent parameter estimates as a metric to quantify the convergence progress. The metric in the ith iteration ($i \geq \rho$) is defined as

$$\Delta_i = \frac{1}{K-1} \sum_{k=1}^{K-1} \text{var}\{\hat{\tau}_k^{((i-\rho+1):i)}\}, \tag{4.26}$$

where i is the iteration index, ρ is the window length and a positive integer, and $\tau_k^{((i-\rho+1):i)}$ is the sequence of estimates of τ_k from the $(i-\rho+1)$th iteration to the ith iteration. We claim the entrance into a neighborhood of convergence if the metric Δ_i ($i \geq \rho$) falls below a predefined threshold η.

We denote this variant of SAEM as SAEM*.

4.4.3 Complexity analysis

Each SAEM* iteration consists of particle filtering, particle smoothing, stochastic approximation, and maximization. For convenience, we assume that a fixed Monte Carlo sample size M is used across iterations.

The time complexity of particle filtering is $O(KNM)$, and that of particle smoothing is $O(KNM^2)$, which includes the computation needed for byproducts. Stochastic approximation takes a time complexity of $O(KN)$, and maximization takes $O(KN)$ for each iteration of the gradient projection method. Therefore, the time complexity of each SAEM* iteration is dominated by that of particle smoothing with $O(KNM^2)$.

The space complexity of particle filtering is $O(KNM)$, and that of particle smoothing is $O(KNM^2)$. Both stochastic approximation and maximization take an extra space complexity of $O(KN)$. Therefore, the space complexity of each SAEM* iteration is also dominated by that of particle smoothing with $O(KNM^2)$.

4.5 Joint estimation with an unknown temporal order

In Sections 4.3 and 4.4, we designed a method to estimate relative clock offsets from sensor observations, given the temporal order of sensors. However, the temporal order is not always known prior to the estimation of relative clock offsets, and thus also needs to be estimated from observations. In this section, we estimate the temporal order via hypothesis testing in a distributed manner.

4.5.1 Maximum likelihood hypothesis

We consider each possible temporal order ξ as a hypothesis, and let Ξ be the set of all candidate hypotheses. We find the most probable hypothesis $\xi \in \Xi$ given sensor observations $\boldsymbol{Y}$, i.e., the maximum a posteriori (MAP) hypothesis [69]

$$\xi_{\text{MAP}} = \arg\max_{\xi \in \Xi} P(\xi|\boldsymbol{Y}) = \arg\max_{\xi \in \Xi} f(\boldsymbol{Y}|\xi)P(\xi). \tag{4.27}$$

In our case, we have no prior preference over any temporal order. Thus, $P(\xi)$ is constant for all $\xi \in \Xi$, and the MAP hypothesis reduces to a maximum likelihood (ML) hypothesis [69]

$$\xi_{\mathrm{ML}} = \arg\max_{\xi \in \Xi} f(\boldsymbol{Y}|\xi). \tag{4.28}$$

4.5.2 Distributed implementation

For a network of K unsynchronized sensors, the cardinality of Ξ is $K!$, so the ML hypothesis approach will not scale on the whole network, since it needs to evaluate every single hypothesis. We thus propose to divide the whole network into groups of size two, and infer local temporal orders from local sensor observations. For a group of two sensors, there exist only two possible temporal orders, each corresponding to a different sign of the relative clock offset between these two sensors, so the ML hypothesis approach is totally tractable.

After obtaining an estimate of the temporal order, we apply the joint estimation method designed in Sections 4.3 and 4.4 to each local group of sensor observations, obtain an estimate of the exact value (including the sign) of the relative clock offset, and fuse local estimates into a global estimate.

To obtain an estimate of the network synchronization status from local estimates, we cannot divide the network into arbitrary groups. Let each sensor be a vertex in a graph, and add an undirected edge between two sensors, if they belong to the same group. The thus created graph has to be connected [70], in order to provide a global estimate. The minimum number of groups is $K - 1$, when the corresponding graph is a path that sequentially visits the K vertices; the maximum number of groups is $K(K-1)/2$, when the corresponding graph is

complete. Since each group corresponds to a distributed task, the quadratic upper bound on the number of groups implies tractability of the distributed implementation.

4.5.3 Hypothesis testing

Consider a group that consists of S_i and S_j, and then $\Xi = \{\tau_{i,j} > 0, \tau_{i,j} < 0\}$. Denote local observations as $\boldsymbol{Y}_{\text{loc}} = \{\boldsymbol{y}_{n,k} : k = i \text{ or } j\}$ and locally observed states as $\boldsymbol{X}_{\text{loc}} = \{\boldsymbol{x}_{n,k} : k = i \text{ or } j\}$. Then, the ML hypothesis is $\arg\max_{\xi\in\Xi} f(\boldsymbol{Y}_{\text{loc}}|\xi)$, which is equivalent to comparing $f(\boldsymbol{Y}_{\text{loc}}|\tau_{i,j} > 0)$ and $f(\boldsymbol{Y}_{\text{loc}}|\tau_{i,j} < 0)$. Since ξ, $\tau_{i,j}$, $\boldsymbol{X}_{\text{loc}}$, and $\boldsymbol{Y}_{\text{loc}}$ form a Markov chain, $f(\boldsymbol{Y}_{\text{loc}}|\xi)$ can be evaluated via the Chapman-Kolmogorov equation as

$$f(\boldsymbol{Y}_{\text{loc}}|\xi) = \iint f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}}) f(\boldsymbol{X}_{\text{loc}}|\tau_{i,j}) f(\tau_{i,j}|\xi) \mathrm{d}\boldsymbol{X}_{\text{loc}} \mathrm{d}\tau_{i,j}. \tag{4.29}$$

We use the Monte Carlo method to compute (4.29). First, we generate a sample of $\tau_{i,j}$, given the hypothesis ξ. Since we have no prior preference over any value of $\tau_{i,j}$, we consider it to be uniformly distributed. Also, since $|\tau_{i,j}| < 1$, $\tau_{i,j}$ is uniformly distributed over $(0, 1)$ if $\xi =$ "$\tau_{i,j} > 0$" or $(-1, 0)$ if $\xi =$ "$\tau_{i,j} < 0$". Then, conditioned on the sample of $\tau_{i,j}$, we generate a sample of $\boldsymbol{X}_{\text{loc}}$ according to the state transition equation, and evaluate $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ using the sample of $\boldsymbol{X}_{\text{loc}}$ according to the observation equation. We repeat the previous steps until we have enough samples. Finally, we compute an average of $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ over all the samples under ξ, and use the average to approximate $f(\boldsymbol{Y}_{\text{loc}}|\xi)$.

In practice, the Monte Carlo method may be difficult to realize. First, the sample space may be so large that most of the generated samples of $\boldsymbol{X}_{\text{loc}}$ are far from the true $\boldsymbol{X}_{\text{loc}}$ and thus show no significant difference in $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ under different hypotheses, which makes it less effective to compare the average. Second, $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ is likely to be extremely close

to zero for most samples. We cannot simply round these numbers to zero, because we need them not for their absolute values but for comparison; we cannot store them in logarithms, either, because we need to compute their average. Hence, the task could be challenging for an ordinary digital processor. However, as we can expect, although poor samples can be generated under either hypothesis, good samples are more likely to come from the true hypothesis than from the false hypothesis. Therefore, we compare the maximum, instead of the average, of $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ under different hypotheses. In this way, we can simply store $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ in its logarithmic form.

The randomness of sampling could lead to errors in hypothesis testing, although with an extremely small probability for a sufficiently large number of samples. In case an error occurs, the joint estimation method that follows hypothesis testing can help validate the correctness of the inferred temporal order and detect the inference error. The particle filter involved in every iteration of the joint estimation method is very sensitive to small $f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,k})$, which is used to update the weight of a particle across iterations, and can easily collapse if most particles are of small weight [34]. Since $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ is the product of $f(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,k})$'s, the collapse of the particle filter also implies a small $f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ and thus suggests the alternative hypothesis. The extra validation further reduces the originally very small probability of error.

4.5.4 Data fusion

A group of S_i and S_j provides a local estimate of $\tau_{i,j}$ for data fusion. Since $\tau_{i,j} = \tau_{i,1} - \tau_{j,1}$, stacking these equations gives us the following equation system:

$$\boldsymbol{A}\boldsymbol{\tau}_{\text{ref}=1} = \boldsymbol{d}, \tag{4.30}$$

where $\boldsymbol{A} \in \mathbb{R}^{L\times(K-1)}$, $\boldsymbol{\tau}_{\text{ref}=1} = [\tau_{2,1}, \tau_{3,1}, \ldots, \tau_{K,1}]^T$, and $\boldsymbol{d} \in \mathbb{R}^{L\times 1}$. Each row of $\boldsymbol{A}$ corresponds to a distributed task, and L is thus the number of distributed tasks. When L takes its lower bound $K-1$, $\boldsymbol{A}$ is a square matrix, and (4.30) can be solved using Gaussian elimination; when $L > K-1$, (4.30) is overdetermined, and can be solved using least squares. With the solution $\hat{\boldsymbol{\tau}}_{\text{ref}=1}$ ready, each $\tau_{i,j}$ can be estimated as $\hat{\tau}_{i,1} - \hat{\tau}_{j,1}$, and a global estimate of the network synchronization status, which includes the global temporal order, can be easily obtained. The central processor then estimates the target trajectory based on sensor observations, given the network synchronization status.

Note that estimates provided by different distributed tasks could conflict. Their values could not agree, e.g., $\tau_{i,j} + \tau_{j,k} \neq \tau_{i,k}$, which is normal, since these estimates are obtained from noisy measurements; their orders could not agree, e.g., $\tau_{i,j} > 0, \tau_{j,k} > 0, \text{but } \tau_{i,k} < 0$, which is extremely unlikely under the dual inference introduced in Section 4.5.3 but still theoretically possible. For value conflicts, ordinary least squares suffices; for order conflicts, robust regression with outlier detection can be used. Also, increasing the number of distributed tasks can add to accuracy and robustness to errors.

4.5.5 Complexity analysis

The number of groups L is $O(K^2)$. For each group, relative clock offset estimation takes a time complexity of $O(NM^2)$, where M denotes the Monte Carlo sample size, and hypothesis testing takes $O(NM)$, if we assume its sample size to be a linear function of the Monte Carlo sample size. The space complexity of hypothesis testing is $O(N)$, and that of relative clock offset estimation is $O(NM^2)$, so the overall space complexity for each group is $O(NM^2)$.

The fusion center takes a time complexity of $O(LK^2 + K^3)$ for data fusion and a time complexity of $O(KNM^2)$ for global target trajectory estimation. The data fusion step takes a space complexity of $O(LK + K^2)$, and the global estimation step takes a space complexity of $O(KNM^2)$.

4.6 Performance analysis

In signal processing, a covariance matrix is often used to evaluate the statistical performance of an estimation method, and is commonly obtained from the inverse of the Fisher information matrix. In this section, we explore how to apply this approach to the EM algorithm under a state-space framework. We develop an algorithm to compute the EM covariance matrix, with complementary derivation given in the Appendix.

4.6.1 Information matrix

The observed, missing, and complete information matrices can be expressed as

$$\mathcal{I}_{\mathrm{o}}(\boldsymbol{\tau}) = -\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \frac{\partial^2}{\partial \boldsymbol{\tau} \partial \boldsymbol{\tau}^T} \log f(\boldsymbol{Y}; \boldsymbol{\tau}), \tag{4.31}$$

$$\mathcal{I}_{\mathrm{m}}(\boldsymbol{\tau}) = -\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \frac{\partial^2}{\partial \boldsymbol{\tau} \partial \boldsymbol{\tau}^T} \log f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}), \text{and} \tag{4.32}$$

$$\mathcal{I}_{\mathrm{c}}(\boldsymbol{\tau}) = -\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \frac{\partial^2}{\partial \boldsymbol{\tau} \partial \boldsymbol{\tau}^T} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau}), \tag{4.33}$$

respectively. Among them, the observed information matrix $\mathcal{I}_{\mathrm{o}}$ is the most relevant, because its inverse is exactly the covariance matrix of the EM estimate. However, it is also the most difficult to compute, because the observed likelihood $f(\boldsymbol{Y}; \boldsymbol{\tau})$ is intractable, which is the very

reason why we use the EM algorithm to circumvent it. A possible approach to calculating $\mathcal{I}_{\mathrm{o}}$ is to use the missing information principle [27]:

$$\mathcal{I}_{\mathrm{o}}(\boldsymbol{\tau}) = \mathcal{I}_{\mathrm{c}}(\boldsymbol{\tau}) - \mathcal{I}_{\mathrm{m}}(\boldsymbol{\tau}). \tag{4.34}$$

The complete information matrix $\mathcal{I}_{\mathrm{c}}$ can be calculated as

$$\mathcal{I}_{\mathrm{c}}(\boldsymbol{\tau}) = -\left[\frac{\partial^2}{\partial \boldsymbol{\tau}_{\mathrm{o}} \partial \boldsymbol{\tau}_{\mathrm{o}}^T} Q_{\mathrm{p}}(\boldsymbol{\tau}_{\mathrm{o}}; \boldsymbol{\tau})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}}, \tag{4.35}$$

which is tractable based on our previous knowledge. The derivation of (4.35) can be found in the Appendix.

The missing information matrix $\mathcal{I}_{\mathrm{m}}$ can be calculated using Louis's formular [57] as

$$\mathcal{I}_{\mathrm{m}}(\boldsymbol{\tau}) = \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[S_{\mathrm{c}}(\boldsymbol{\tau}) S_{\mathrm{c}}^T(\boldsymbol{\tau})\right] - S_{\mathrm{o}}(\boldsymbol{\tau}) S_{\mathrm{o}}^T(\boldsymbol{\tau}), \tag{4.36}$$

where $S_{\mathrm{c}}(\boldsymbol{\tau}) = \frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau})$ is the complete score function, and, correspondingly, $S_{\mathrm{o}}(\boldsymbol{\tau}) = \frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{Y}; \boldsymbol{\tau})$ is the observed score function.

In (4.36), $S_{\mathrm{o}}(\boldsymbol{\tau})$ can be computed as

$$S_{\mathrm{o}}(\boldsymbol{\tau}) = \left[\frac{\partial}{\partial \boldsymbol{\tau}_{\mathrm{o}}} Q_{\mathrm{p}}(\boldsymbol{\tau}_{\mathrm{o}}; \boldsymbol{\tau})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}}, \tag{4.37}$$

whose derivation can be found in the Appendix, and $S_{\mathrm{c}}(\boldsymbol{\tau})$ can be computed as

$$\begin{aligned} S_{\mathrm{c}}(\boldsymbol{\tau}) =& \frac{\partial}{\partial \boldsymbol{\tau}} \left[\sum_{n=1}^{N-1} \log f(\boldsymbol{x}_{n+1,1} | \boldsymbol{x}_{n,K}; \tau_K) + \sum_{n=1}^{N} \sum_{k=1}^{K-1} \log f(\boldsymbol{x}_{n,k+1} | \boldsymbol{x}_{n,k}; \tau_k) \right] \\ =& \begin{bmatrix} s_1 & s_2 & \cdots & s_{K-1} \end{bmatrix}^T, \end{aligned} \tag{4.38}$$

where

$$\begin{aligned} s_k =& \frac{\partial}{\partial \tau_k} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau}) \\ =& \sum_{n=1}^{N-1} \frac{\partial}{\partial \tau_k} \log f(\boldsymbol{x}_{n+1,1} | \boldsymbol{x}_{n,K}; \tau_K) + \sum_{n=1}^{N} \frac{\partial}{\partial \tau_k} \log f(\boldsymbol{x}_{n,k+1} | \boldsymbol{x}_{n,k}; \tau_k) \end{aligned} \tag{4.39}$$

for $k = 1, 2, \cdots, K-1$. Hence,

$$\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \left[S_{\mathrm{c}}(\boldsymbol{\tau}) S_{\mathrm{c}}^T(\boldsymbol{\tau}) \right] = \{ \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} [s_i s_j] \}_{i,j}, \tag{4.40}$$

where $\{ \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} [s_i s_j] \}_{i,j}$ denotes a matrix whose i,jth element is $\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} [s_i s_j]$. The matrix, or equivalently $\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \left[S_{\mathrm{c}}(\boldsymbol{\tau}) S_{\mathrm{c}}^T(\boldsymbol{\tau}) \right]$, is the very term that makes Louis's method considered to be intractable [60]. However, as we will see, it can be computed under our problem formulation, and the approach can also be extended to general state-space models.

4.6.2 Algorithm design

To compute each $\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} [s_i s_j]$, it suffices to know the joint conditional density of every two pairs of adjacent states. We propose a recursive and parallel algorithm to compute the joint conditional densities of all possible combinations of state pairs.

Let $(\boldsymbol{x}_a, \boldsymbol{x}_{a+1})$ and $(\boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1})$, where $0 \leq k \leq KN-2$ and $1 \leq a \leq KN-k-1$, be two arbitrary pairs of adjacent states.

If $k = 0$, then $(\boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}) = (\boldsymbol{x}_a, \boldsymbol{x}_{a+1})$, and the joint conditional density is exactly the byproduct of smoothing that we obtain in Section 4.3.3:

$$f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y}; \boldsymbol{\tau}) = f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau}), \tag{4.41}$$

which is calculated as a byproduct of smoothing.

If $k = 1$, then $(\boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}) = (\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2})$ and the joint conditional density can be computed as

$$f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y}; \boldsymbol{\tau}) = \frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})} f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau}), \tag{4.42}$$

where $f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau})$ was obtained when $k = 0$. Please see the Appendix for detailed derivation.

If $k = 2$, then $(\boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}) = (\boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3})$ and the joint conditional density can be computed as

$$f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y}; \boldsymbol{\tau}) = \frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})} f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau}), \tag{4.43}$$

where $f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau})$ was obtained when $k = 1$. Please see the Appendix for detailed derivation.

If $k = k' \geq 3$, then the joint conditional density can be computed as

$$f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1}|\boldsymbol{Y};\boldsymbol{\tau}) = \frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y};\boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1}|\boldsymbol{Y};\boldsymbol{\tau})} \times \int f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1}|\boldsymbol{Y};\boldsymbol{\tau})\mathrm{d}\boldsymbol{x}_{a+2}, \quad (4.44)$$

where $f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1}|\boldsymbol{Y};\boldsymbol{\tau})$ was obtained when $k = k' - 1$. Please see the Appendix for derivation.

Following (4.41)–(4.44), we are able to compute $f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y};\boldsymbol{\tau})$ for every possible combination of a and k and thus every matrix element in $\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[S_{\mathrm{c}}(\boldsymbol{\tau})S_{\mathrm{c}}^T(\boldsymbol{\tau})\right]$.

Note that for a fixed k, $f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y};\boldsymbol{\tau})$ with different a's can be computed in parallel, since each of them only depends on results with smaller k's. This leads to a significant run-time saving. Also, the algorithm is completely compatible with Monte Carlo approximations.

4.6.3 Covariance matrix

The covariance matrix of the EM estimate $\hat{\boldsymbol{\tau}}$ is obtained from the inverse of the observed information matrix $\mathcal{I}_{\mathrm{o}}(\boldsymbol{\tau})$ evaluated at $\boldsymbol{\tau} = \hat{\boldsymbol{\tau}}$, i.e.,

$$\mathrm{cov}(\hat{\boldsymbol{\tau}}) = \mathcal{I}_{\mathrm{o}}(\hat{\boldsymbol{\tau}})^{-1} = \left(\mathcal{I}_{\mathrm{c}}(\hat{\boldsymbol{\tau}}) - \mathcal{I}_{\mathrm{m}}(\hat{\boldsymbol{\tau}})\right)^{-1}, \quad (4.45)$$

where $\mathcal{I}_{\mathrm{c}}(\hat{\boldsymbol{\tau}})$ is calculated according to (4.35), and $\mathcal{I}_{\mathrm{m}}(\hat{\boldsymbol{\tau}})$ according to Louis's formula in (4.36) together with our algorithm.

In [71], it is shown that $\hat{\boldsymbol{\tau}}$ is a stationary point of $L(\boldsymbol{\tau};\boldsymbol{Y})$ and thus $\log L(\boldsymbol{\tau};\boldsymbol{Y})$, i.e.,

$$S_{\mathrm{o}}(\hat{\boldsymbol{\tau}}) = \left[\frac{\partial}{\partial \boldsymbol{\tau}} \log L(\boldsymbol{\tau};\boldsymbol{Y})\right]_{\boldsymbol{\tau}=\hat{\boldsymbol{\tau}}} = 0. \tag{4.46}$$

Therefore, according to (4.36), $\mathcal{I}_{\mathrm{m}}(\hat{\boldsymbol{\tau}})$ can be calculated as

$$\mathcal{I}_{\mathrm{m}}(\hat{\boldsymbol{\tau}}) = \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\hat{\boldsymbol{\tau}}}\left[S_{\mathrm{c}}(\hat{\boldsymbol{\tau}}) S_{\mathrm{c}}^{T}(\hat{\boldsymbol{\tau}})\right]. \tag{4.47}$$

4.7 Numerical examples

In this section, we use numerical examples to demonstrate the performance of the proposed methods. We show first the accuracy of the joint estimation method with a known temporal order, then the effectiveness of the hypothesis testing approach to temporal ordering, and finally the effect that an unknown temporal order has on the accuracy of joint estimation. We compare the proposed joint estimation method with an ordinary sequential target estimation method to demonstrate the advantages of taking synchronization into account, and with the augmented state-space method to show the advantages of the utilized joint estimation strategy. Also, we compare the simulation results with the ground truth to show the accuracy of the proposed solution.

We tested our methods on wireless sensor networks with different synchronization statuses and different number of sensors in the following numerical examples.

The target observed by sensors in a network followed a noisy constant velocity kinematic model in a 2-dimensional space with the transition function

$$\boldsymbol{g}(\boldsymbol{x}_{n,k}, \tau) = \boldsymbol{D}(\tau) \cdot \boldsymbol{x}_{n,k}, \tag{4.48}$$

where

$$\boldsymbol{x}_{n,k} = \begin{bmatrix} x_{n,k}^{(1)} \\ x_{n,k}^{(2)} \\ \dot{x}_{n,k}^{(1)} \\ \dot{x}_{n,k}^{(2)} \end{bmatrix}, \quad \boldsymbol{D}(\tau) = \begin{bmatrix} 1 & 0 & \tau & 0 \\ 0 & 1 & 0 & \tau \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.49}$$

and the transition noise

$$\boldsymbol{u}_{n,k}(\tau) \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{R}(\tau)\right), \tag{4.50}$$

where

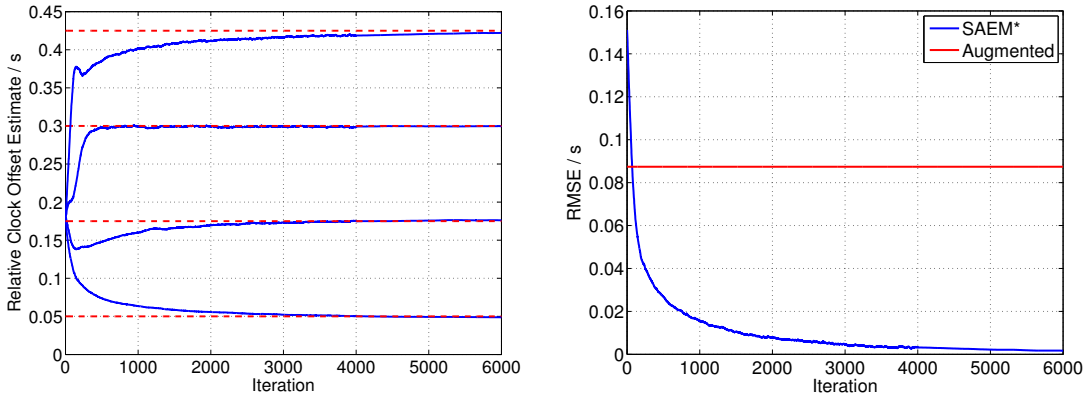
$$\boldsymbol{R}(\tau) = \sigma_u^2 \begin{bmatrix} \frac{\tau^3}{3} & 0 & \frac{\tau^2}{2} & 0 \\ 0 & \frac{\tau^3}{3} & 0 & \frac{\tau^2}{2} \\ \frac{\tau^2}{2} & 0 & \tau & 0 \\ 0 & \frac{\tau^2}{2} & 0 & \tau \end{bmatrix}. \tag{4.51}$$

S_k was located at $\boldsymbol{l}_k = (l_k^{(1)}, l_k^{(2)})$ with its observation function

$$h_k(\boldsymbol{x}_{n,k}) = \sqrt{(x_{n,k}^{(1)} - l_k^{(1)})^2 + (x_{n,k}^{(2)} - l_k^{(2)})^2}, \tag{4.52}$$

and its observation noise

$$v_{n,k} \sim \mathcal{N}(0, \sigma_v^2). \tag{4.53}$$



(a) (b)

Figure 4.1: Clock synchronization results for a network of 5 sensors: (a) convergence of $\{\boldsymbol{\tau}^{(i)}\}$; (b) RMSE of $\boldsymbol{\tau}^{(i)}$.

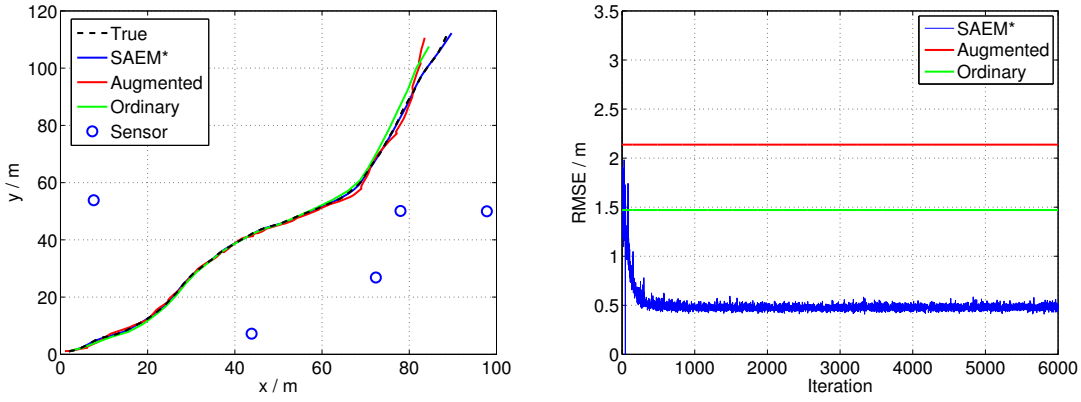
For all the numerical examples to be presented, we assume $N = 30$, $\sigma_u = 0.71$, $\sigma_v = 0.1$, $\rho = 500$, and $\eta = 1 \times 10^{-5}$. Also, we assume $\boldsymbol{x}_0 = [0\text{m}, 0\text{m}, 1\text{m/s}, 1\text{m/s}]^T$ and

$$\boldsymbol{x}_{1,1} = \boldsymbol{D}(1) \cdot \boldsymbol{x}_0 + \boldsymbol{u}_{1,1}(1), \tag{4.54}$$

i.e. the prior information about the distribution of the initial state $\boldsymbol{x}_{1,1}$ is $\mathcal{N}\left(\boldsymbol{D}(1) \cdot \boldsymbol{x}_0, \boldsymbol{R}(1)\right)$.

4.7.1 Clock synchronization

Fig. 4.1 shows the clock synchronization results of a network of 5 sensors located at on a 2D plane, with relative clock offsets $\boldsymbol{\tau} = [0.0500\text{s}, 0.1750\text{s}, 0.3000\text{s}, 0.4250\text{s}]^T$. Prior to estimation, $\boldsymbol{\tau}$ was unknown, but the temporal order $S_1 \to S_2 \to S_3 \to S_4 \to S_5$ was known. SAEM* with a fixed sample size of 800 and an initial guess $\hat{\boldsymbol{\tau}}^{(0)} = [0.1800\text{s}, 0.1800\text{s}, 0.1800\text{s}, 0.1800\text{s}]^T$ was used.



(a) (b)

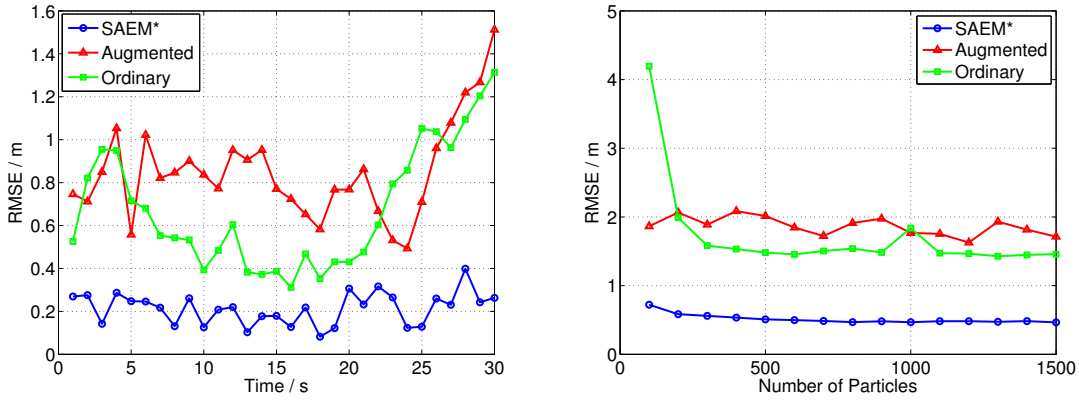
Figure 4.2: Sequential target estimation results for a network of 5 sensors: (a) trajectory estimates; (b) the trajectory estimation RMSEs across iterations.

As shown in Fig. 4.1(a), the sequence of estimates approached $\boldsymbol{\tau}$ rapidly and then slowed down while stably converging to $\boldsymbol{\tau}$, which is exactly how SAEM* works. The final estimate is $\hat{\boldsymbol{\tau}} = [0.0488\text{s}, 0.1761\text{s}, 0.2997\text{s}, 0.4221\text{s}]^T$ with a root-mean-square error (RMSE) equal to 0.0016s. Fig. 4.1(b) shows how the RMSE dropped from 0.1511s to 0.0016s as the algorithm converged, in comparison with the RMSE of the estimate provided by the augmented state-space method with the same sample size.

4.7.2 Target estimation

In Fig. 4.2, we demonstrate the sequential target estimation results, which came together with the clock synchronization results in Section 4.7.1, in comparison with the augmented state-space method and the ordinary target estimation method.

Fig. 4.2(a) compares estimates of the target trajectory. As shown in the figure, the trajectory estimate provided by the proposed method almost coincided with the ground truth, while

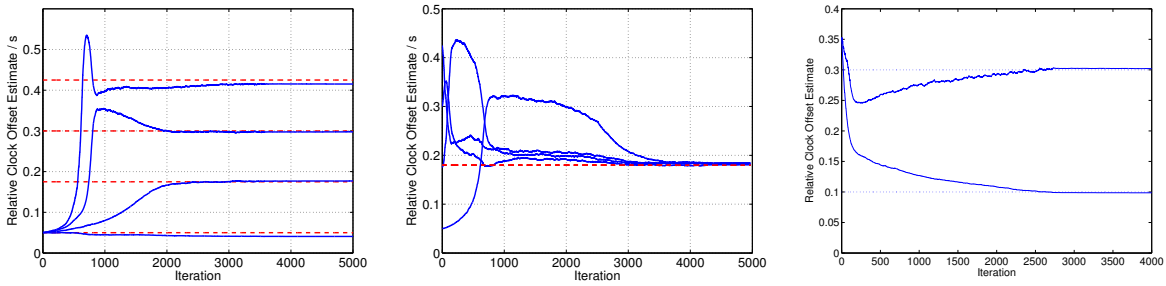


(a) (b)

Figure 4.3: Sequential target estimation results for a network of 5 sensors: (a) state estimation RMSE as a function of time; (b) trajectory estimation RMSE as a function of the Monte Carlo sample size.

those provided by the augmented state-space method and the ordinary method started to deviate from the ground truth halfway down their trajectories.

Fig. 4.2(b) shows that the RMSE of the trajectory estimate provided by the proposed method decreased rapidly with slight fluctuation and then stayed stable around 0.5m, which is significantly lower than those of the other two methods. The fluctuation results from the randomness of Monte Carlo sampling, and will attenuate as the sample size becomes larger. Compared with Fig. 4.1(b), it is not difficult to notice that the target state estimates reached convergence earlier than the relative clock offset estimates. This implies that the target estimation part in the proposed method is relatively robust to errors of the relative clock offset estimates within a certain range. Also, as shown in the figure, initial estimates of the proposed method have higher RMSEs than that of the ordinary method, which implies that a poor guess of the relative clock offsets can be even worse than no guess at all.



(a) (b) (c)

Figure 4.4: Clock synchronization results using SAEM* with 800 particles

In Fig. 4.3, we show how target estimation errors changed as time went on and as the number of particles varied.

Fig. 4.3(a) compares the RMSE as a function of time. Data in Fig. 4.3(a) were obtained from repeated experiments with a fixed number of particles, 800. As shown in the figure, the RMSE of the proposed method was lower than those of the other two methods at every time point. Also, the RMSE of the proposed method stayed at the same level as time went on, while those of the other two methods tended to increase with time because of the cumulation of estimation errors.

Fig. 4.3(b) compares the RMSE of a trajectory estimate as a function of the number of particles. Data in Fig. 4.3(b) were also obtained from repeated experiments. It is obvious that the RMSE of the proposed method was lower than those of the other methods under any sample size. The RMSE of the proposed method slightly decreased as the sample size grew, which agrees with the fact that the accuracy of a Monte Carlo approximation grows with the sample size. However, since the decrease was not significant, it might not be cost effective to use a large sample size (e.g., more than 1000 particles). A medium sample size (e.g., 500–1000 particles) should be sufficient in terms of both approximation accuracy

and computational efficiency. The RMSE of the augmented state-space method tended to decrease as the number of particles increased; that of the ordinary method was fairly high when the sample size was small, but dropped and stayed at the same level, when the sample size was large.

4.7.3 Convergence

Fig. 4.4 shows three examples of the convergence of the clock synchronization part of the proposed joint estimation method (with a known temporal order) on different networks with different synchronization statuses and different initial guesses. In Fig. 4.4(a), we set $\boldsymbol{\tau} = [0.0500\text{s}, 0.1750\text{s}, 0.3000\text{s}, 0.4250\text{s}]^T$, $\hat{\boldsymbol{\tau}}^{(0)} = [0.0500\text{s}, 0.0500\text{s}, 0.0500\text{s}, 0.0500\text{s}]^T$, and $\hat{\boldsymbol{\tau}} = [0.0408\text{s}, 0.1770\text{s}, 0.2977\text{s}, 0.4152\text{s}]^T$; in Fig. 4.4(b), we set $\boldsymbol{\tau} = [0.1800\text{s}, 0.1800\text{s}, 0.1800\text{s}, 0.1800\text{s}]^T$, $\hat{\boldsymbol{\tau}}^{(0)} = [0.0500\text{s}, 0.1750\text{s}, 0.3000\text{s}, 0.4250\text{s}]^T$, and $\hat{\boldsymbol{\tau}} = [0.1828\text{s}, 0.1848\text{s}, 0.1806\text{s}, 0.1825\text{s}]^T$; in Fig. 4.4(c), we set $\boldsymbol{\tau} = [0.1000\text{s}, 0.3000\text{s}]^T$, $\hat{\boldsymbol{\tau}}^{(0)} = [0.3500\text{s}, 0.3500\text{s}]^T$, and $\hat{\boldsymbol{\tau}} = [0.0986\text{s}, 0.3020\text{s}]^T$.

The proposed method achieved comparable accuracy in all three examples and demonstrated its robustness to different network synchronization statuses. Fig. 4.4(a) and Fig. 4.1(a) are based on the same network with the same synchronization status but different initial guesses for the EM algorithm, and their convergence results demonstrated the robustness of the proposed method to different initial guesses.

Fig. 4.5 compares the convergence performance of SAEM*, SAEM, and MCEM with the same initial guess but different numbers of particles. Results were obtained from a network of 3 sensors located at $(30\text{m}, 10\text{m})$, $(10\text{m}, 20\text{m})$, and $(40\text{m}, 20\text{m})$ with relative clock offsets $\boldsymbol{\tau} = [0.3\text{s}, 0.1\text{s}]^T$.

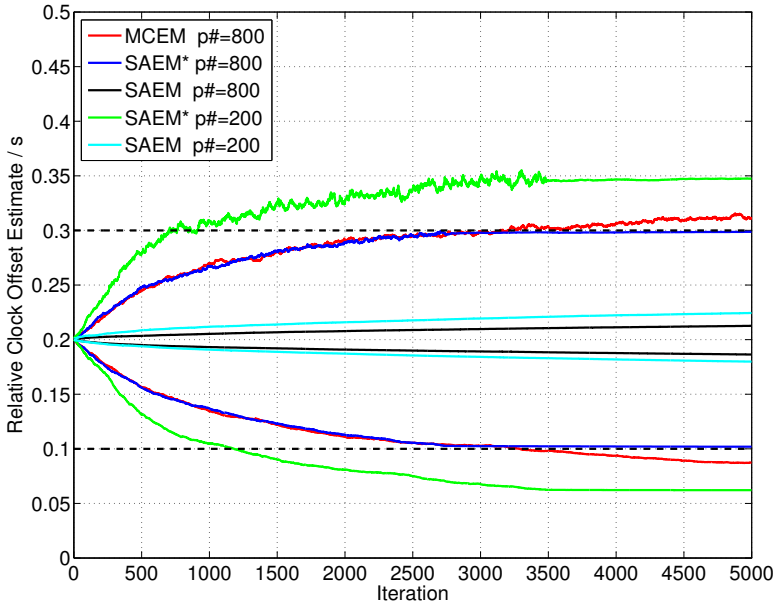


Figure 4.5: A comparison of the convergence performance of MCEM with 800 particles, SAEM* with 800 particles, SAEM with 800 particles, SAEM* with 200 particles, and SAEM with 200 particles, with the same initial guess $\hat{\boldsymbol{\tau}}^{(0)} = [0.2\text{s}, 0.2\text{s}]^T$ and the same network of sensors located at $(30\text{m}, 10\text{m})$, $(10\text{m}, 20\text{m})$, and $(40\text{m}, 20\text{m})$ with relative clock offsets $\boldsymbol{\tau} = [0.3\text{s}, 0.1\text{s}]^T$.

As shown in Fig. 4.5, MCEM with 800 particles slowed down as it approached $\boldsymbol{\tau}$, but then continued to fluctuate somewhere beyond $\boldsymbol{\tau}$ due to the limited sample size; SAEM* with 800 particles behaved similarly to MCEM with 800 particles before the moving average started, and stably converged to $\boldsymbol{\tau}$ thanks to the moving average; SAEM with 800 particles approached $\boldsymbol{\tau}$ at a considerably slow pace, because the moving average started from the first iteration, and would thus take considerably many iterations to converge. As stated in Section 4.4.2, SAEM, as well as SAEM*, is guaranteed to converge with any finite number of particles. Hence, we also tested SAEM and SAEM* on a small sample size. As shown in the Fig. 4.5, SAEM with 200 particles converged slightly faster than SAEM with 800 particles, but still much slower than the other methods; SAEM* with 200 particles approached $\boldsymbol{\tau}$ the

fastest among all the tested methods, but got stabilized somewhere far beyond $\boldsymbol{\tau}$, due to the accumulated approximation errors caused by a small sample size. Note that SAEM* with 200 particles would still converge to $\boldsymbol{\tau}$, since its second stage is equivalent to SAEM with 200 particles starting with a different initial guess, but might take many iterations, as SAEM often does. In conclusion, SAEM* with a medium sample size is the optimal choice for our problem.

4.7.4 Temporal ordering

We assumed no knowledge of the temporal order, and applied the distributed hypothesis testing method proposed in Section 4.5 to the same wireless sensor network studied in Section 4.7.1 with a known temporal order. We divided the network into (S_1, S_3), (S_2, S_4), (S_2, S_5), (S_3, S_4), and (S_4, S_5), and found the maximum likelihood temporal order of each pair through hypothesis testing with a Monte Carlo sample size of 10^6 for each candidate hypothesis.

Table 4.1 summarizes the hypothesis testing results obtained from these distributed tasks. For each row, which corresponds to a distributed task, we found the maximum of $\log f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ among samples of $\boldsymbol{X}_{\text{loc}}$ under either hypothesis (listed in the 3rd and 4th columns), and chose the hypothesis with a higher $\max \log f(\boldsymbol{Y}_{\text{loc}}|\boldsymbol{X}_{\text{loc}})$ to be our decision. As we can see, all the decisions were correctly made, with no need for further verification by the following joint

Table 4.1: Hypothesis testing results

i	j	$\tau_{i,j} > 0$	$\tau_{i,j} < 0$	Decision
1	3	-1.1445×10^4	-1.2454×10^4	$\tau_{1,3} > 0$
2	4	-8.1943×10^3	-1.5929×10^4	$\tau_{2,4} > 0$
2	5	-1.1046×10^4	-3.7557×10^4	$\tau_{2,5} > 0$
3	4	-6.3506×10^3	-1.1820×10^4	$\tau_{3,4} > 0$
4	5	-8.3623×10^3	-1.1749×10^4	$\tau_{4,5} > 0$

estimation method. Moreover, the difference between the 3rd and 4th columns in each row is fairly large (no less than 10^3). Note that the difference is between logarithms, which implies that the ratio without taking the logarithm would be considerably large (no less than e^{1000}). Such a ratio would make the leading hypothesis overwhelmingly dominant in the comparison, thus further lowering the probability of error. The ratio also verifies the effectiveness and reliability of the proposed hypothesis testing method in return.

With local temporal orders known, we applied the joint estimation method to each distributed task, and estimated the relative clock offset based on local observations. Fig. 4.6 shows the clock synchronization result of each distributed task. Based on these local estimates, we obtained through least squares $\hat{\boldsymbol{\tau}}_{\text{ref}=1} = [-0.0439\text{s}, -0.2196\text{s}, -0.5168\text{s}, -0.9389\text{s}]^T$, determined from $\hat{\boldsymbol{\tau}}_{\text{ref}=1}$ the global temporal order $S_1 \to S_2 \to S_3 \to S_4 \to S_5$, and converted $\hat{\boldsymbol{\tau}}_{\text{ref}=1}$ to

$$\begin{aligned}\hat{\boldsymbol{\tau}} &= [\hat{\tau}_{1,2}, \hat{\tau}_{2,3}, \hat{\tau}_{3,4}, \hat{\tau}_{4,5}]^T \\ &= [0.0439\text{s}, 0.1757\text{s}, 0.2972\text{s}, 0.4221\text{s}]^T.\end{aligned}$$

The RMSE of $\hat{\boldsymbol{\tau}}$ is 0.0037s, comparable to, although slightly higher than, that of the estimate given by the joint estimation method with a known temporal order. Here, a slightly higher RMSE is reasonable because of the distributed processing enforced by the absence of knowledge of the global temporal order. Based on $\hat{\boldsymbol{\tau}}$, we obtained a target trajectory estimate with an average RMSE of 0.5195m from repeated experiments, which is comparable to that obtained with a known temporal order as shown in Fig. 4.2(b).

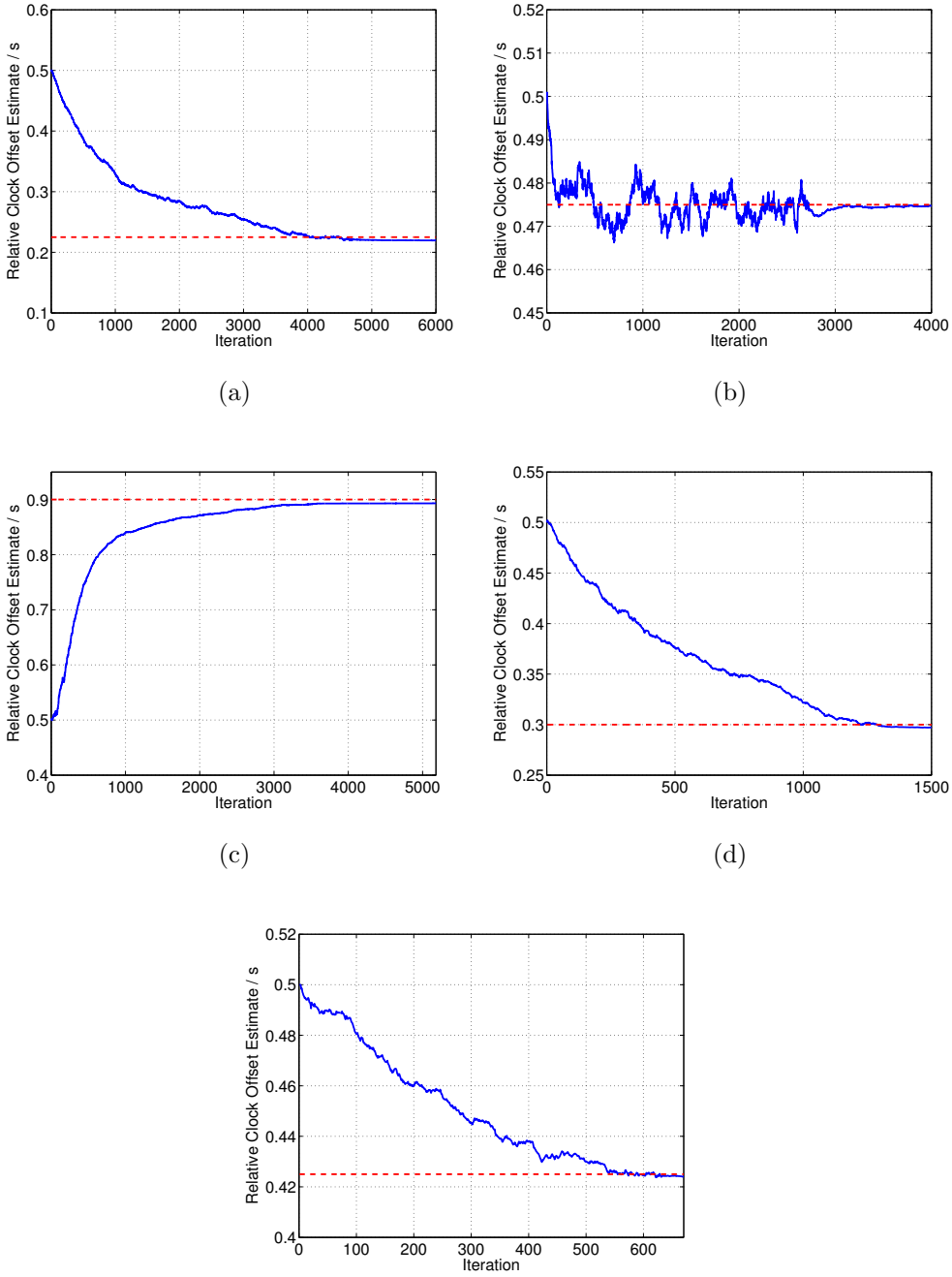


Figure 4.6: Relative clock offset estimation results from distributed tasks: (a) for (S_1, S_3), $\tau_{1,3} = 0.2250$s, $\hat{\tau}_{1,3} = 0.2196$s; (b) for (S_2, S_4), $\tau_{2,4} = 0.4750$s, $\hat{\tau}_{2,4} = 0.4748$s; (c) for (S_2, S_5), $\tau_{2,5} = 0.9000$s, $\hat{\tau}_{2,5} = 0.8932$s; (d) for (S_3, S_4), $\tau_{3,4} = 0.3000$s, $\hat{\tau}_{3,4} = 0.2972$s; (e) for (S_4, S_5), $\tau_{4,5} = 0.4250$s, $\hat{\tau}_{4,5} = 0.4240$s.

4.7.5 Covariance matrix results

In addition, we tested the covariance matrix algorithm on the numerical example in Section 4.7.1, and obtained the following result,

$$\begin{bmatrix} 0.0010 & -0.0005 & -0.0055 & 0.0076 \\ -0.0005 & 0.0279 & -0.0009 & -0.0300 \\ -0.0055 & -0.0009 & 0.1784 & -0.2098 \\ 0.0076 & -0.0300 & -0.2098 & 0.2844 \end{bmatrix} \times 10^{-5},$$

which is positive definite and coincides with the fact that most variables are negatively correlated (because their sum is constrained). For comparison, we also computed the sample covariance matrix,

$$\begin{bmatrix} 0.0235 & -0.0201 & -0.0024 & -0.0507 \\ -0.0201 & 0.0650 & -0.0172 & 0.0402 \\ -0.0024 & -0.0172 & 0.0283 & -0.0117 \\ -0.0507 & 0.0402 & -0.0117 & 0.1695 \end{bmatrix} \times 10^{-5},$$

which has the same order of magnitude as the asymptotic covariance matrix. The small variance and covariance in both matrices also imply that the EM algorithm has stable estimation performance.

4.8 Chapter summary

In this chapter, we proposed to jointly estimate sequential target states and network synchronization status based on sensor observations. We developed a centralized joint estimation method under the assumption of a known temporal order, proposed an hypothesis testing method to learn an unknown temporal order from asynchronous sensor observations, and derived a distributed joint estimation method based on the previous two methods to work with an unknown temporal order. We demonstrated the performance of the proposed method through numerical examples. The numerical examples showed that the trajectory estimation error when the synchronization problem was considered was 34% lower than that when the synchronization problem was ignored. The numerical examples also proved four points: 1) the joint estimation method converges to the true synchronization status and outputs an accurate target trajectory estimate, given a known temporal order, 2) the hypothesis testing method outputs the true temporal order, 3) the distributed method outputs an estimate of the network synchronization status with negligible sacrifice in accuracy resulting from the absence of prior knowledge of the temporal order, and 4) clock synchronization significantly improves the accuracy of sequential target estimation. Since the proposed observation-based clock synchronization method makes use of the already obtained and distributed sensor observations and thus avoids extra communications of timestamps, we also concluded that it saves more energy for a wireless sensor network than traditional timestamp-based clock synchronization methods.

The proposed joint estimation framework is inspired by clock synchronization problems but can be used to solve other parameter estimation problems under a state-space model. Also, the proposed covariance matrix algorithm presents a numerical approach to computing the covariance matrix of the EM algorithm.

Chapter 5

Conclusions and Future Work

5.1 Summary and conclusions

In this dissertation we studied distributed target tracking using wireless sensor networks and sensor synchronization for target tracking.

We first considered distributed particle filtering based on distributed fusion of local posteriors provided by local particle filters. We derived an optimal distributed fusion rule from Bayes' theorem and implemented it via average consensus. We proved the convergence of the proposed distributed fusion rule and applied it to local posteriors parametrically represented as Gaussian mixtures. We designed a weighted mixture importance sampling approach to the nonlinear fusion of Gaussian mixtures. Using numerical examples, we showed that the proposed distributed particle filtering algorithm is significantly more accurate than other posterior-based algorithms and that it is competitive in accuracy with likelihood-based and particle-based algorithms. We also demonstrated the communication efficiency of the proposed algorithm, achieved by the compactness of Gaussian mixture models. We further discussed the advantages of the proposed algorithm beyond accuracy and efficiency, such as

the flexibility of each individual sensor in its sensing, processing, and parametric representation.

We next considered a flexible parametric representation for local posteriors in the proposed distributed particle filtering algorithm. We designed a hierarchical clustering algorithm, combined with the EM algorithm, to learn from weighted Monte Carlo samples a Gaussian mixture model consisting of an adaptively determined number of mixture components. We designed an adaptive splitting strategy for hierarchical clustering and sent the clustering result to the EM algorithm as an informed initial guess. We showed that adaptive splitting in hierarchical clustering improves the accuracy of hierarchical clustering and also reduces the number of EM iterations needed to find a maximum likelihood solution, thus achieving computational efficiency.

Finally, we considered the synchronization problem of wireless sensor networks and its impact on target tracking. We developed a statistical method to infer the unknown relative clock offsets between sensors from their periodic observations of a common target. We formulated the synchronization problem as a joint estimation problem for both the unknown relative clock offsets and the hidden target states, and solved the problem using a stochastic variant of the EM algorithm. We discussed the performance of the stochastic EM algorithm under Monte Carlo approximations, and developed an approximation to the covariance matrix of an EM estimate through Monte Carlo approximations. We showed that the proposed synchronization method converges to the ground truth and that sensor synchronization improves the accuracy of target tracking.

5.2 Future directions

In this section, we point out potential future research directions.

Analytical fusion of Gaussian mixtures: In this dissertation, we solved the nonlinear Gaussian mixture fusion problem through importance sampling, which often requires a large number of samples and thus can be computationally intensive. It would be preferable if we could fuse Gaussian mixtures analytically, thus avoiding sampling and improving both computational efficiency and stability.

Approximation to powers of Gaussian mixtures: In nonlinear fusion of Gaussian mixtures, we often need to compute a power of a Gaussian mixture. The exponent can be a positive integer, a positive fraction, or a negative number. Importance sampling might be a numerical solution, but it is again computationally intensive and sometimes inaccurate due to the choice of the proposal distribution. For this reason, it would be interesting to find an analytical approximation to the power of a Gaussian mixture. Moreover, it would also help the analytical fusion of Gaussian mixtures.

Distributed fusion based on randomized gossip: In this dissertation, we used average consensus for distributed fusion. Randomized gossip is another framework for distributed fusion. In randomized gossip, each iteration often involves only two neighboring sensors, thus possibly simplifying the challenge in nonlinear fusion of Gaussian mixtures. Also, randomized gossip is an asynchronous protocol and thus easier to implement in practice than average consensus.

Distribtued multiple particle filtering: Multiple particle filtering [72] is a particle filtering strategy that effectively deals with high-dimensional systems. Existing methods of

multiple particle filtering all work under a centralized implementation and do not scale as the wireless sensor network grows. It would be interesting to study a distributed implementation of multiple particle filtering.

Online sensor synchronization: The sensor synchronization method considered in the dissertation is an offline method. It would be interesting to explore an online solution to the same problem, so that we can sequentially update our estimate of the relative clock offsets based on incoming information only.

Bibliography

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

[2] D. Üstebay, M. Coates, and M. Rabbat, "Distributed auxiliary particle filters using selective gossip," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, Prague, Czech Republic, May 2011, pp. 3296–3299.

[3] C. J. Bordin and M. G. S. Bruno, "Consensus-based distributed particle filtering algorithms for cooperative blind equalization in receiver networks," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, Prague, Czech Republic, May 2011, pp. 3968–3971.

[4] C. J. Bordin and M. G. Bruno, "Distributed particle filtering for blind equalization in receiver networks using marginal non-parametric approximations," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*, Florence, Italy, May 2014, pp. 7984–7987.

[5] S. Farahmand, S. Roumeliotis, and G. B. Giannakis, "Set-membership constrained particle filter: Distributed adaptation for sensor networks," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4122–4138, June 2011.

[6] V. Savic, H. Wymeersch, and S. Zazo, "Belief consensus algorithms for fast distributed target tracking in wireless sensor networks," *Signal Processing*, vol. 95, pp. 149–160, Feb. 2014.

[7] O. Hlinka, F. Hlawatsch, and P. M. Djurić, "Likelihood consensus-based distributed particle filtering with distributed proposal density adaptation," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 2012, pp. 3869–3872.

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

[9] O. Sluciak, 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 *45th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, Nov. 2011, pp. 1766–1770.

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

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

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

[13] D. Gu, "Distributed particle filter for target tracking," in *2007 IEEE International Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, pp. 3856–3861.

[14] C. M. Bishop, in *Pattern Recognition and Machine Learning*. Springer, 2007, ch. 9.

[15] P. Chavali and A. Nehorai, "Managing multi-modal sensor networks using price theory," *IEEE Transactions on Signal Processing*, vol. 60, no. 9, pp. 4874–4887, June 2012.

[16] R. V. D. Merwe, A. Doucet, N. D. Freitas, and E. Wan, "The unscented particle filter," in *NIPS*, vol. 2000, Aug. 2000, pp. 584–590.

[17] D. Guo and X. Wang, "Dynamic sensor collaboration via sequential Monte Carlo," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 6, pp. 1037–1047, Aug. 2004.

[18] Z. Yan, B. Zheng, and J. Cui, "Distributed particle filter for target tracking in wireless sensor network," in *14th European Signal Processing Conference*, Florence, Italy, Sep. 2006, pp. 1–5.

[19] O. Hlinka, P. M. Djurić, and F. Hlawatsch, "Time-space-sequential distributed particle filtering with low-rate communications," in *43rd Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, Nov. 2009, pp. 196–200.

[20] X. Sheng, Y.-H. Hu, and P. Ramanathan, "Distributed particle filter with GMM approximation for multiple targets localization and tracking in wireless sensor network," in *4th International Symposium on Information Processing in Sensor Networks*, Los Angeles, CA, Apr. 2005, pp. 181–188.

[21] M. G. Bruno and S. S. Dias, "Collaborative emitter tracking using Rao-Blackwellized random exchange diffusion particle filtering," *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, pp. 1–18, Feb. 2014.

[22] 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.

[23] 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.

[24] J. Li and A. Nehorai, "Joint sequential target estimation and clock synchronization in wireless sensor networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 74–88, June 2015.

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

[26] L. Xiao, S. Boyd, and S. Lall, "Distributed average consensus with time-varying Metropolis weights," *Automatica*, 2006.

[27] 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.

[28] N. R. Ahmed and M. Campbell, "Fast consistent Chernoff fusion of Gaussian mixtures for ad hoc sensor networks," *IEEE Transactions on Signal Processing*, vol. 60, no. 12, pp. 6739–6745, Nov. 2012.

[29] P. Paclík and J. Novovicová, "Number of components and initialization in Gaussian mixture model for pattern recognition," in *Artificial Neural Nets and Genetic Algorithms*. Springer, 2001, pp. 406–409.

[30] T. Huang, H. Peng, and K. Zhang, "Model selection for Gaussian mixture models," *arXiv preprint arXiv:1301.3558*, 2013.

[31] R. J. Steele and A. Raftery, "Performance of Bayesian model selection criteria for Gaussian mixture models," *Frontiers of Statistical Decision Making and Bayesian Analysis*, pp. 113–130, 2010.

[32] C. Olivier, F. Jouzel, and A. Matouat, "Choice of the number of component clusters in mixture models by information criteria," in *Vision Interface*, 1999, pp. 74–81.

[33] A. T. Ihler, E. B. Sudderth, W. T. Freeman, and A. S. Willsky, "Efficient multiscale sampling from products of Gaussian mixtures," *Advances in Neural Information Processing Systems*, vol. 16, pp. 1–8, 2004.

[34] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, "Obstacles to high-dimensional particle filtering," *Monthly Weather Review*, vol. 136, no. 12, pp. 4629–4640, Dec. 2008.

[35] Y. Bar-Shalom, P. K. Willett, and X. Tian, in *Tracking and Data Fusion: A Handbook of Algorithms*. YBS Publishing, 2011, ch. 1.

[36] 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, Feb. 2002.

[37] W. Li and Y. Jia, "Consensus-based distributed multiple model UKF for jump Markov nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 227–233, Dec. 2012.

[38] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, 2012.

[39] J. R. Hershey and P. Olsen, "Approximating the Kullback Leibler divergence between Gaussian mixture models," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, Honolulu, HI, Apr. 2007, pp. IV–317–320.

[40] H. Akaike, "Information theory and an extension of the maximum likelihood principle," *Selected Papers of Hirotugu Akaike*, pp. 199–213, 1998.

[41] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[42] A. Moore, "A tutorial on kd-trees," *Technical Report, University of Cambridge Computer Laboratory*, 1991.

[43] D. Boley, "Principal direction divisive partitioning," *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 325–344, 1998.

[44] J. Li and A. Nehorai, "Distributed particle filtering via optimal fusion of Gaussian mixtures," in *18th International Conference on Information Fusion*, Washington D.C., July 2015, pp. 1182–1189.

[45] A. T. Ihler, J. W. Fisher III, and A. S. Willsky, "Using sample-based representations under communications constraints," Lab. Inform. and Decision Syst., Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. 2601, 2005.

[46] Z. Fan, E. Liu, and B. Xu, "Weighted principal component analysis," in *Artificial Intelligence and Computational Intelligence*. Springer, 2011, pp. 569–574.

[47] A. W. Bowman and A. Azzalini, *Applied Smoothing Techniques for Data Analysis*. Oxford University Press, 1997.

[48] O. Hlinka, F. Hlawatsch, and P. M. Djurić, "Distributed particle filtering in agent networks," *IEEE Signal Processing Magazine*, vol. 1, no. 30, pp. 61–81, Jan. 2013.

[49] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed, "Detection, classification, and tracking of targets," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 17–29, Mar. 2002.

[50] P. M. Djurić, M. Vemula, and M. F. Bugallo, "Target tracking by particle filtering in binary sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2229–2238, June 2008.

[51] O. Ozdemir, R. Niu, and P. K. Varshney, "Tracking in wireless sensor networks using particle filtering: Physical layer considerations," *IEEE Transctions on Signal Processing*, vol. 57, no. 5, pp. 1987–1999, 2009.

[52] J. Beaudeau, M. F. Bugallo, and P. M. Djurić, "Target tracking with asynchronous measurements by a network of distributed mobile agents," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, 2012, pp. 3857–3860.

[53] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. ACM, 2003, pp. 138–149.

[54] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[55] J. Liu and M. West, "Combined parameter and state estimation in simulation-based filtering," *Sequential Monte Carlo Methods in Practice*, pp. 197–223, 2001.

[56] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, 2nd ed. Wiley-Interscience, 2008.

[57] T. A. Louis, "Finding the observed information matrix when using the EM algorithm," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 44, no. 2, pp. 226–233, 1982.

[58] I. Meilijson, "A fast improvement to the EM algorithm on its own terms," *Journal of the American Statistical Association*, vol. 51, no. 1, pp. 127–138, 1989.

[59] J. B. Carlin, *Seasonal Analysis of Economic Time Series*. Harvard University, 1987.

[60] X.-L. Meng and D. B. Rubin, "Using EM to obtain asymptotic variance-covariance matrices: The SEM algorithm," *Journal of the American Statistical Association*, vol. 86, no. 416, pp. 899–909, Dec. 1991.

[61] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009.

[62] L. Murray, "Bayesian learning of continuous time dynamical systems with applications in functional magnetic resonance imaging," Ph.D. dissertation, School of Informatics, University of Edinburgh, 2008.

[63] D. J. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. Springer, 2002.

[64] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Oxford Handbook of Nonlinear Filtering*, 2011.

[65] G. Wei and M. A. Tanner, "A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms," *Journal of the American Statistical Association*, vol. 85, no. 411, pp. 699–704, Sep. 1990.

[66] S. F. Nielsen, "The stochastic EM algorithm: Estimation and asymptotic results," *Bernoulli*, vol. 6, no. 3, pp. 457–489, June 2000.

[67] J. G. Booth, J. P. Hobert, and W. Jank, "A survey of Monte Carlo algorithms for maximizing the likelihood of a two-stage hierarchical model," *Statistical Modelling*, vol. 1, pp. 333–349, 2001.

[68] J. G. Booth and J. P. Hobert, "Maixmizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm," *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, vol. 61, no. 1, pp. 265–285, 1999.

[69] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[70] D. B. West, *Introduction to Graph Theory*, 2nd ed. Pearson, 2000.

[71] W. Jank, "Stochastic variants of the EM algorithm: Monte Carlo, quasi-Monte Carlo and more," in *Proceedings of the American Statistical Association*, Minneapolis, Minnesota, 2005.

[72] P. M. Djuric, T. Lu, and M. F. Bugallo, "Multiple particle filtering," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 2007, pp. III–1181–1184.

Appendix A

Derivation of Equations

Derivation of the forward part (4.14) in Section 4.3.3:

$$
\begin{aligned}
f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)}) &= \frac{f(\boldsymbol{x}_n,\boldsymbol{y}_n|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)})}{f(\boldsymbol{y}_n|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)})} \\
&\propto f(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) f(\boldsymbol{x}_n|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) \\
&\propto f(\boldsymbol{y}_n|\boldsymbol{x}_n) \int f(\boldsymbol{x}_n,\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) \mathrm{d}\boldsymbol{x}_{n-1} \\
&\propto f(\boldsymbol{y}_n|\boldsymbol{x}_n) \int f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1},\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) \times f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) \mathrm{d}\boldsymbol{x}_{n-1} \\
&\propto f(\boldsymbol{y}_n|\boldsymbol{x}_n) \int f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1};\hat{\boldsymbol{\tau}}^{(i)}) \times f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:(n-1)};\hat{\boldsymbol{\tau}}^{(i)}) \mathrm{d}\boldsymbol{x}_{n-1}.
\end{aligned}
$$

Derivation of the backward part (4.16) in Section 4.3.3:

$$
\begin{aligned}
f(\boldsymbol{x}_n|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)}) &= \int f(\boldsymbol{x}_n,\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1} \\
&= \int f(\boldsymbol{x}_n|\boldsymbol{x}_{n+1},\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1} \\
&= \int \frac{f(\boldsymbol{x}_{n+1},\boldsymbol{x}_n|y_{1:n};\hat{\boldsymbol{\tau}}^{(i)})}{f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})} f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1} \\
&= \int \frac{f(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n,y_{1:n};\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})}{f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})} f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1} \\
&= \int \frac{f(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n;\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})}{\int f(\boldsymbol{x}_{n+1}|\boldsymbol{x}_n;\hat{\boldsymbol{\tau}}^{(i)})f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_n} f(\boldsymbol{x}_{n+1}|\boldsymbol{y}_{1:KN};\hat{\boldsymbol{\tau}}^{(i)})\mathrm{d}\boldsymbol{x}_{n+1}.
\end{aligned}
$$

Calculation of the complete information matrix (4.35) in Section 4.6.1:

$$
\begin{aligned}
\mathcal{I}_{\mathrm{c}}(\boldsymbol{\tau}) &= -\,\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \frac{\partial^2}{\partial\boldsymbol{\tau}\partial\boldsymbol{\tau}^T}\log f(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{\tau}) \\
&= -\int f(\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau})\left[\frac{\partial^2}{\partial\boldsymbol{\tau}_o\partial\boldsymbol{\tau}_{\mathrm{o}}^T}\log f(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{\tau}_{\mathrm{o}})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}}\mathrm{d}\boldsymbol{X} \\
&= -\left[\frac{\partial^2}{\partial\boldsymbol{\tau}_{\mathrm{o}}\partial\boldsymbol{\tau}_{\mathrm{o}}^T}\int f(\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau})\log f(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{\tau}_{\mathrm{o}})\mathrm{d}\boldsymbol{X}\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}} \\
&= -\left[\frac{\partial^2}{\partial\boldsymbol{\tau}_{\mathrm{o}}\partial\boldsymbol{\tau}_{\mathrm{o}}^T}\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\log f(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{\tau}_{\mathrm{o}})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}} \\
&= -\left[\frac{\partial^2}{\partial\boldsymbol{\tau}_{\mathrm{o}}\partial\boldsymbol{\tau}_{\mathrm{o}}^T}Q_{\mathrm{p}}(\boldsymbol{\tau}_{\mathrm{o}};\boldsymbol{\tau})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}}.
\end{aligned}
$$

Calculation of the observed score function (4.37) in Section 4.6.1:

$$
\begin{aligned}
S_{\mathrm{o}}(\boldsymbol{\tau}) &= \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}[S_{\mathrm{c}}(\boldsymbol{\tau})] \\
&= \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[\frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau})\right] \\
&= \left[\frac{\partial}{\partial \boldsymbol{\tau}_{\mathrm{o}}} \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau}_{\mathrm{o}})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}} \\
&= \left[\frac{\partial}{\partial \boldsymbol{\tau}_{\mathrm{o}}} Q_{\mathrm{p}}(\boldsymbol{\tau}_{\mathrm{o}}; \boldsymbol{\tau})\right]_{\boldsymbol{\tau}_{\mathrm{o}}=\boldsymbol{\tau}},
\end{aligned}
$$

where the first step is proved below:

$$
\begin{aligned}
\mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}[S_{\mathrm{c}}(\boldsymbol{\tau})] =& \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[\frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\tau})\right] \\
=& \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[\frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) + \frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{Y}; \boldsymbol{\tau})\right] \\
=& \mathbb{E}_{\boldsymbol{X}|\boldsymbol{Y};\boldsymbol{\tau}}\left[\frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau})\right] + \frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{Y}; \boldsymbol{\tau}) \\
=& \int f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) \frac{\partial}{\partial \boldsymbol{\tau}} \log f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{X} + S_{\mathrm{o}}(\boldsymbol{\tau}) \\
=& \int f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) \frac{\frac{\partial}{\partial \boldsymbol{\tau}} f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau})} \mathrm{d}\boldsymbol{X} + S_{\mathrm{o}}(\boldsymbol{\tau}) \\
=& \int \frac{\partial}{\partial \boldsymbol{\tau}} f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{X} + S_{\mathrm{o}}(\boldsymbol{\tau}) \\
=& \frac{\partial}{\partial \boldsymbol{\tau}} \int f(\boldsymbol{X}|\boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{X} + S_{\mathrm{o}}(\boldsymbol{\tau}) \\
=& S_{\mathrm{o}}(\boldsymbol{\tau}).
\end{aligned}
$$

Derivation of the EM covariance matrix algorithm in Section 4.6.2:

For $k = 1$, (4.42) results from

$$\begin{aligned}
&f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a|\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a|\boldsymbol{x}_{a+1}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&\frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})} f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}|\boldsymbol{Y}; \boldsymbol{\tau}).
\end{aligned}$$

For $k = 2$, (4.43) results from

$$\begin{aligned}
&f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k}, \boldsymbol{x}_{a+k+1}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a|\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&f(\boldsymbol{x}_a|\boldsymbol{x}_{a+1}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau}) \\
=&\frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1}|\boldsymbol{Y}; \boldsymbol{\tau})} f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+3}|\boldsymbol{Y}; \boldsymbol{\tau}).
\end{aligned}$$

For $k = k' \geq 3$, (4.44) results from

$$
\begin{aligned}
& f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1} | \boldsymbol{Y}; \boldsymbol{\tau}) \\
&= \int f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1} | \boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{x}_{a+2} \\
&= \int f(\boldsymbol{x}_a | \boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1} | \boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{x}_{a+2} \\
&= \int f(\boldsymbol{x}_a | \boldsymbol{x}_{a+1}, \boldsymbol{Y}; \boldsymbol{\tau}) f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1} | \boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{x}_{a+2} \\
&= \frac{f(\boldsymbol{x}_a, \boldsymbol{x}_{a+1} | \boldsymbol{Y}; \boldsymbol{\tau})}{f(\boldsymbol{x}_{a+1} | \boldsymbol{Y}; \boldsymbol{\tau})} \int f(\boldsymbol{x}_{a+1}, \boldsymbol{x}_{a+2}, \boldsymbol{x}_{a+k'}, \boldsymbol{x}_{a+k'+1} | \boldsymbol{Y}; \boldsymbol{\tau}) \mathrm{d}\boldsymbol{x}_{a+2}.
\end{aligned}
$$

Vita

Jichuan Li

Degrees

Ph.D., Electrical Engineering, Washington University in St. Louis, Missouri, USA, May 2016
M.S., Electrical Engineering, Washington University in St. Louis, Missouri, USA, December 2014
B.S., Electrical Engineering, Fudan University, Shanghai, China, July 2011

Professional Memberships

The Institute of Electrical and Electronics Engineers (IEEE)
IEEE Signal Processing Society

Publications

Journal Publications:

J. Li and A. Nehorai, "Distributed particle filtering via optimal fusion of Gaussian mixtures," submitted.

J. Li and A. Nehorai, "Adaptive Gaussian mixture learning in distributed particle filtering," submitted.

J. Li and A. Nehorai, "Joint sequential target estimation and clock synchronization in wireless sensor networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 74–88, June 2015.

S. Krishnan, B. Kumfer, W. Wu, **J. Li**, A. Nehorai, and R. Axelbaum, "An approach to thermocouple measurements that reduces uncertainties in high temperature," *Energy & Fuels*, vol. 29, no. 5, pp. 3446–3455, Apr. 2015.

Conference Publications:

J. Li and A. Nehorai, "Joint sequential target state estimation and clock synchronization in wireless sensor networks," in *48th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, Nov., 2014, pp. 525–529.

J. Li and A. Nehorai, "Distributed particle filtering via optimal fusion of Gaussian mixtures," in *18th International Conference on Information Fusion*, Washington D.C., July 2015, pp. 1182–1189.

J. Li and A. Nehorai, "Adaptive Gaussian mixture learning in distributed particle filtering," in *6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Cancun, Mexico, Dec. 2015, pp. 221–224.

May 2016