Distributed Particle Filters for Sensor Networks

Mark Coates

Department of Electrical and Computer Engineering,
McGill University
3480 University St, Montreal, Quebec, Canada H3A 2A7
coates@ece.mcgill.ca,
WWW home page: http://www.ece.mcgill.ca/∼coates

Abstract. This paper describes two methodologies for performing distributed particle filtering in a sensor network. It considers the scenario in which a set of sensor nodes make multiple, noisy measurements of an underlying, time-varying state that describes the monitored system. The goal of the proposed algorithms is to perform on-line, distributed estimation of the current state at multiple sensor nodes, whilst attempting to minimize communication overhead. The first algorithm relies on likelihood factorization and the training of parametric models to approximate the likelihood factors. The second algorithm adds a predictive scalar quantizer training step into the more standard particle filtering framework, allowing adaptive encoding of the measurements. As its primary example, the paper describes the application of the quantization-based algorithm to tracking a manoeuvring object. The paper concludes with a discussion of the limitations of the presented technique and an indication of future avenues for enhancement.

1 Introduction

The goal of many sensor networks is to detect and track changes in the monitored environment. Such scenarios arise in target tracking [1,9], in time-varying density estimation, and in the task of robot navigation [3, 4]. In these situations, the class of signal (or information) processing task that must be collaboratively performed by the sensor nodes migrates from static estimation or detection to on-line estimation, filtering or change-point detection. It becomes important to design sequential algorithms that can dynamically fuse the information recorded by the sensors without requiring excessive exchange of either data or algorithmic information.

In this paper, we consider the situation where the nature of the monitored environment can be captured by a Markovian state-space model that involves potentially nonlinear dynamics, nonlinear observations, and non-Gaussian innovation and observation noises. Our goal is to perform sequential estimation of the current system state at multiple sensor nodes in the network. In nonlinear and non-Gaussian scenarios, the decentralized Kalman filter [2], which admits an attractive, relatively low-dimensional parametric information exchange between sensor nodes, becomes inapplicable. Extended Kalman filters, grid-based methods and Gaussian-sum filters are possible alternatives [5, 6], but these all have limitations, and information exchange is not as simple. The class of sequential Monte Carlo (or particle filtering) methods [7] is attractive because of its power and flexibility. These methods keep track of a set of "particles", or candidate state descriptions. The methods evaluate how well each particle conforms to the dynamic model and explains the observations, using this assessment to generate a weighted particulate approximation to the filtering distribution, and hence form state estimates.

There are two causes for concern in the adoption of sequential Monte Carlo algorithms in sensor networks. First, the algorithms are substantially more computationally demanding than more parametric alternatives. Second, the development of decentralized or distributed particle filters has been limited, so it is somewhat unclear what information must be exchanged in order to implement a collaborative algorithm. The second concern is more pressing than the first; a sensor node with reasonable processing power and memory is likely to be able to cope with the computational demands of a particle filter unless state changes are very rapid, in which case sensing and network communication also begin to be very difficult exercises. Identifying the information that needs to be exchanged between nodes is more difficult.

It is certainly undesirable to transmit the raw (or finely quantized) data to a set of processing network nodes. The communication cost is high and substantial sensor node energy is consumed. On the other hand, the natural information representation within a particle filter is a set of particles and associated weights. The exchange of these in raw form almost certainly involves the transmission of many more bits than the exchange of the raw data. In some cases, it is possible to develop a parametric approximation of the particle-based filtering distribution (a method adopted in [8, 9]). We explore this idea further in this paper, examining some of the restrictions in model structure that this approach implies.

In this paper, we propose two distributed particle filtering algorithms for sensor networks. The first approach is based on factorizing the likelihood, and forming parametric approximations to products of likelihood factors (using the particles and their associated likelihoods as training data). The model parameters are then exchanged between sensor nodes, instead of the data or exact particle information. This approach places restrictions on the structure of the problem, because it must be possible to both factorize the likelihood and develop reasonably accurate, low-dimensional parametric models to describe the factors. The approach results in substantial communication savings when the data dimension is much higher than the dimension of the parameter space of the models approximating the likelihood-factors.

The second distributed algorithm uses an adaptive data-encoding approach. It involves the training of predictive linear quantizers at every time-step based on a common particle filter maintained at all nodes. Sensor nodes transmit the quantized data to one another; the compression can be substantial because the particle filter can provide a very good indication of where a sensor measurement is likely to be. This approach places no restrictions on the nature of the likelihood function, but the training of optimal linear quantizers is computationally expensive, so the amount of data that can be processed at each time-step is limited by the computational power of the sensor nodes. With this limitation in mind, we describe a hierarchical sensor network framework involving two classes of nodes, A and B. Class A nodes are more computationally powerful, and have more energy resources. In the framework we describe, all computation is performed by class A nodes, and all sensing by class B nodes. The class A nodes manage the class B nodes, selecting only a small set to make measurements at any time step. The adaptive data-encoding approach is then feasible.

The paper is organized as follows. Section 2 states the problem and reviews the core steps of centralized particle filtering algorithms. Section 3 describes the two distributed particle filtering algorithms. Section 3 describes the two distributed particle filtering algorithms. Section 4 outlines a hierarchical sensor network framework that uses the adaptive-encoding based filtering algorithm. Section 5 reports on simulations in which the hierarchical sensor network was used to track an object manouevring through a sensor field. Finally, Section 6 discusses the proposed algorithms, indicating limitations and proposing avenues for improvement and further research.

1.1 Related Work

There have been some efforts to design distributed Bayes (or particle) filters in the sensor networks [8, 9] and in the artificial intelligence community [4]. The work in [8] is targeted at tracking an object or several objects. In the single object scenario, one leader node is active at any time instant. The leader node maintains a belief state, effectively a filtering distribution (represented either parametrically or through a particle set). Based on its belief state, the leader node evaluates the expected utility of neighbouring sensors and chooses the node with highest utility to become the new leader node for the next time step. It then propagates its belief state to the new leader node, either by exchanging parameters or by transmitting particle locations and weights. In follow-up work, the authors have investigated techniques for approximating the particle distribution and transmitting this approximation.

The work in [4] designs distributed particle filters for decentralized data fusion. The aim is to use local particle filters to determine which measurements are worth sharing. The scheme works using a query-response system. Each sensor node maintains a local particle filter. Neighbouring nodes query one another for useful sensor measurements; a query is comprised of a small set of randomly-selected particles with entire state trajectories. Based on this set of particles, the queried node examines its set of unshared measurements (its own or those received from other sensors), and transmits only the most informative measure, as evaluated by some form of divergence measure.

The distributed particle filters we describe in this paper differ in purpose and implementation. The key difference is that we strive to maintain a *common* particle representation of the posterior distribution at multiple nodes in the network at every time instant. This means that the manner in which the measured data at any time instant is utilised must be consistent across the network.

2 Generalized Problem Statement and Centralized Approaches

In this paper, we are concerned with the problem of performing on-line state estimation for multi-dimensional signals that can be modelled using Markovian state-space models that are (potentially) nonlinear and non-Gaussian. The unobserved global state $\{\mathbf{x}_t; t \in \mathbb{N}\}$ is modelled as a Markov process with initial distribution $p(\mathbf{x}_0)$ and transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. The observations $\{\mathbf{y}_t; t \in \mathbb{N}^*\}$ are assumed to be conditionally independent (in time) given the process $\mathbf{x}_t$ and of marginal distribution $p(\mathbf{y}_t|\mathbf{x}_t)$. We denote by $\mathbf{x}_{0:t} \triangleq \{\mathbf{x}_0, \ldots, \mathbf{x}_t\}$ and by $\mathbf{y}_{1:t} \triangleq \{\mathbf{y}_1, \ldots, \mathbf{y}_t\}$, respectively, the system state and the observations up to time t. The measurements $\mathbf{y}_t$ are recorded by K sensors, and we use $\mathbf{y}_t^k$ to denote the subset of observations made by the k-th sensor.

The aim is to estimate on-line the posterior distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, and functions derived from it, such as the filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t}$ or expectations of the form $I(f_t) = \mathbb{E}_{p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})}[f_t(\mathbf{x}_{0:t})]$.

2.1 Centralized Particle Filtering Approach

In the case where the sensor measurements are available at a central location, several approaches have been proposed to perform the task outlined in the previous section. In the linear, Gaussian state-space case, the Kalman filter provides analytical update expressions for tracking the evolution of the posterior distributions. In the case of nonlinear and/or non-Gaussian models, approximation approaches such as the extended Kalman filter [5] or grid-based methods [6] can be adopted. An alternative approach is to employ one of the algorithms belonging to the class of sequential Monte Carlo (or particle filtering) methods [7]. This section now briefly outlines sequential Monte Carlo methods, drawing on descriptions from [7] and references therein.

Sequential Monte Carlo methods have been dubbed particle filters because they maintain a set of state trajectories (or particles) that are candidate representations of the system state. There is an *importance weight* associated with each particle; at a given time instant, this weight is representative of how well the state trajectory conforms to model dynamics and describes the set of observations, relative to the other particles. Whenever there is a transition between time instants and a new observation becomes available, each trajectory is extended, and its associated weight adjusted according to how well it explains the new observation.

There are three generic steps that appear, in one form or another, in the majority of sequential Monte Carlo methods. Individual algorithms have variations on the outlined steps or have additional steps to improve estimation performance, but the steps form the foundation for the methodology. The first step is the initialisation of N particles, denoted by $\{\mathbf{x}_{0:t}^{(i)}; i = 1, \ldots, N\}$. In this initialisation phase, each particle is sampled from the initial distribution: $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$, and every importance weight is initialised to $w_0^{(i)} = 1/N$. After initialisation, the remaining two steps, the *importance sampling* step and the *selection* step, are repeated at every time instant. First, in the importance sampling step, for each $i = 1, \ldots, N$, $\widetilde{\mathbf{x}}_t^{(i)}$ is sampled from an importance distribution $\pi(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})$, which may be any distribution that has the same support as the posterior. A trajectory proposal is then formed by appending this sample to the existing i-th particle to form $\widetilde{\mathbf{x}}_{0:t}^{(i)} = \left(\mathbf{x}_{0:t-1}^{(i)}, \widetilde{\mathbf{x}}_t^{(i)}\right)$. The importance weights are evaluated according to $\widetilde{w}_t^{(i)} = \frac{p(\mathbf{y}_t|\widetilde{\mathbf{x}}_t^{(i)})p(\widetilde{\mathbf{x}}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{\pi(\widetilde{\mathbf{x}}_t^{(i)}|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})}$. The set of importance weights is normalized to sum to one. In the selection step, N particles $\{\mathbf{x}_{0:t}^{(i)}; i = 1, \ldots, N\}$ are formed by sampling with replacement from the set $\{\widetilde{\mathbf{x}}_{0:t}^{(i)}; i = 1, \ldots, N\}$ where the probability of sampling the i-th trajectory is $\widetilde{w}_t^{(i)}$.

Estimates of the posterior distribution, the filtering distribution, or of expectations are formed based on the N particles $\{\widetilde{\mathbf{x}}_{0:t}^{(i)}; i = 1, \ldots, N\}$ and the associated, normalized weights $\widetilde{w}_t^{(i)}$. The posterior distribution is estimated by the weighted empirical distribution $\widehat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} \widetilde{w}_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$. Expectations $I(f_t)$ are estimated as $\widehat{I}_N(f_t) = \sum_{i=1}^{N} f_t(\mathbf{x}_{0:t}^{(i)})$.

In the centralized particle filtering case, with K sensors, the communication required per time instant (neglecting overhead bits) is $\sum_{k=1}^{K} D_k M_k$ bits. For each sensor k, M_k is the number of communication hops to the fusion centre, and D_k is the number of bits necessary for an adequate representation of its measured data. D_k is dependent on the required accuracy of the tracking function and the encoding mechanism. In the next section, we focus on developing a method that can dramatically reduce the value of D_k (as compared to a naive implementation).

3 Distributed Particle Filters

This section develops two distributed algorithms designed to track the state of a non-linear dynamic system based on noisy measurements made by a set of physically isolated sensors. The algorithms consists of K particle filters, with one filter running at every sensor in the network. It is the nature of the communication between sensors that differs between the two algorithms. At this point, it is useful to define *synchronized* particle filters; in this paper, these are particle filters whose random number generators have been initialized at the same point and which generate the same number of random numbers per time step. This means that their random draws are always the same).

3.1 Dissemination of Raw Data

In order to update its particle approximations to the posterior distributions when new data becomes available at time t, each filter needs to calculate, for each particle $i = 1, \ldots N$, the likelihood function $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$, which determines the new weight of the particle. If the importance sampling distribution is data-dependent, the data is also required for the sampling procedure. The dissemination of quantized data through the network generates a similar communication overhead as the centralized approach, approximately $\sum_{k=1}^{K} D_k M_k$ bits, where D_k is, as before, the number of bits required for adequate data representation, but M_k is now the number of communication hops required to disseminate the data of sensor k throughout the network. In the worst case, M_k should be linear in K, and the communication cost is $\mathcal{O}(KD)$ where $D = \sum_{k=1}^{K} D_k$.

3.2 Factorizable Likelihoods: A Parametric Modelling Approach

It is possible to adopt an alternative approach to data dissemination in the special case where the likelihood function is factorizable, $p(\mathbf{y}_t|\mathbf{x}_t) = \prod_{k=1}^{K} p(\mathbf{y}_t^k|\mathbf{x}_t)$, and each factor $p(\mathbf{y}_t^k|\mathbf{x}_t)$ can be described (or approximated) by a parametric model $\mathcal{F}_k(\mathbf{x}_t; \boldsymbol{\theta}_t^k)$. The model parameters $\boldsymbol{\theta}_t^k$ are estimated from training data pairs $\{\left(\mathbf{x}_t^{(i)}, p(\mathbf{y}_t^k|\mathbf{x}_t^{(i)})\right); i = 1, \ldots, N\}$, and the parameters disseminated instead of the data itself. Unfortunately, unless the data represented by each sensor at each time instant has high dimension, it is likely that an adequate parametric representation of the likelihood function will require more bits than the data itself. The potential advantage of the scheme is that there is the possibility of performing model "aggregation" at each sensor node, thereby avoiding the communication expense engendered by the need to disseminate the entire parameter set $\{\boldsymbol{\theta}_t^k; k = 1, \ldots, K\}$ throughout the sensor network.

We now describe an algorithm that performs a form of model aggregation across sensor nodes. Figure 1 presents the algorithm in high-level pseudo-code format. In this algorithm, we need to make a further assumption on the nature of the likelihood. Not only do we need to be able to approximate individual likelihood factors by a parametric model, we need models $\mathcal{G}_k(\mathbf{x}_t; \boldsymbol{\phi}_t^k)$ that can approximate products of such likelihood factors, $\prod_{j \in S(k)} p(\mathbf{y}_t^j|\mathbf{x}_t)$. Here $S(k)$ is the set of likelihood factors whose product is modelled at node k. Below, we consider the scenario where there is a single communication chain from node 1 to node K, with any node k in the interior of the chain communicating only with nodes $k-1$ and $k+1$. The algorithm is very similar if there is a tree structure for communication; parameters are simply exchanged between parents and children instead of neighbours in the chain.

The particle filter at each node is initialized as in the standard sequential Monte Carlo framework, by sampling N particles from $p(\mathbf{x}_0)$. At time instant t, Node 1 samples from its importance distribution $\pi_1(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t}^1)$ to generate N particles $\{\widetilde{\mathbf{x}}_t^{(i)}; i = 1, \ldots, N\}$. The importance distribution may depend on $\mathbf{y}_{1:t}^1$, but it cannot depend on measurements at other sensors, which are unavailable to node 1. Node 1 calculates the value of its likelihood factor for each one of these particles for the current observation, $p(\mathbf{y}_t^k|\widetilde{\mathbf{x}}_t^{(i)})$, and then trains the model $\mathcal{G}_1(\mathbf{x}_t;\boldsymbol{\phi}_t^1)$ using data pairs from the training set $\{(\widetilde{\mathbf{x}}_t^{(i)}, p(\mathbf{y}_t^1|\widetilde{\mathbf{x}}_t^{(i)})); i = 1,\ldots,N\}$. The values $\boldsymbol{\phi}_t^1$ are then appropriately quantized and transmitted to node 2 in the chain.

Node 2 also extends the trajectories by sampling from its importance distribution $\pi_2(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t}^2, \boldsymbol{\phi}_t^1)$, generating values $\widetilde{\mathbf{x}}_t^{(i)}$. Note that the importance distribution can depend on the transmitted parameters $\boldsymbol{\phi}_t^1$, and the samples may differ from those generated at node 1. Node 2 trains a model $\mathcal{G}_2(\mathbf{x}_t;\boldsymbol{\phi}_t^2)$ to fit the product of likelihood factors $p(\mathbf{y}_t^1|\mathbf{x}_t)p(\mathbf{y}_t^2|\mathbf{x}_t)$. The training is performed according to data pairs from the training set $\{\left(\widetilde{\mathbf{x}}_t^{(i)}, \mathcal{G}_1(\widetilde{\mathbf{x}}_t^{(i)};\boldsymbol{\theta}_t^1)p(\mathbf{y}_t^2|\widetilde{\mathbf{x}}_t^{(i)})\right); i = 1,\ldots,N\}$. The process continues, with node k training a model $\mathcal{G}_k(\mathbf{x}_t;\boldsymbol{\phi}_t^k)$ to fit $\prod_{j=1}^k p(\mathbf{y}_t^j|\mathbf{x}_t)$ using the training data $\{\left(\widetilde{\mathbf{x}}_t^{(i)}, \mathcal{G}_{k-1}(\widetilde{\mathbf{x}}_t^{(i)};\boldsymbol{\phi}_t^{k-1})p(\mathbf{y}_t^k|\widetilde{\mathbf{x}}_t^{(i)})\right); i = 1,\ldots,N\}$. At the K-th sensor node, the parameters of a model $\mathcal{G}_K(\mathbf{x}_t;\boldsymbol{\phi}_t^K)$ have been trained. This model attempts to fit the global likelihood $p(\mathbf{y}_t|\mathbf{x}_t) = \prod_{k=1}^K p(\mathbf{y}_t^k|\mathbf{x}_t)$.

In the next phase of the algorithm, the estimated parameters $\boldsymbol{\phi}_t^K$ are propagated back along the communication chain. Node k uses its knowledge of the model $\mathcal{G}^K$ to calculate estimates of likelihood for its samples $\widetilde{\mathbf{x}}_t^{(i)}$. These estimates are then used to determine importance weights and to perform the resampling step. The algorithm, as described thus far, results in a different set of particles and different estimates at each sensor node. If synchronized particle filters are used, then this effect can be eliminated. Instead of performing resampling based on its initial trajectory extensions, each sensor node generates a new set of trajectory extensions $\widetilde{\mathbf{x}}_t^{(i)}$ according to a common importance distribution $\pi(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)}, \boldsymbol{\phi}_t^K)$. Due to filter synchronization, these particle extensions are common to all nodes, as are the calculated importance weights and the resampling step. The set of particles remains common across all nodes, as do the estimates. This procedure allows importance sampling to take place from a distribution that is dependent on all the data in the network. The originally sampled particle extensions could have been generated in areas where the posterior is insubstantial, due to consideration of only the local data.

The communication cost of this algorithm per time step, neglecting overhead bits, is $KP_K + \sum_{k=1}^K P_k$, where P_k is the number of bits required to represent the parameter set $\boldsymbol{\phi}_k$. Bounding P_k by P, the worst-case for any node k, the cost becomes $\mathcal{O}(KP)$. Comparing this to the cost of disseminating the raw data, which was $\mathcal{O}(KD)$, with D being the number of bits required to represent all the measured data, we see that there is substantial saving if $P \ll D$. This will be the case when there are many sensors, so that the data dimension is high, and when the likelihood factors can be represented by simple models. Each $\mathcal{G}_k$ is a function on the state-space, whose dimension should be substantially smaller than that of the data. Care must be taken in model selection to ensure that the dimension of the parameter space is reasonable and that the parameters can be

Distributed Parametric Approximation Particle Filter

```
1. Initialisation, $t = 0$.
   – For each sensor $k = 1, \ldots, K$
      • For $i = 1, \ldots, N$, sample $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$ and set $t = 1$.

2. First Importance sampling step and forward parameter exchange
   – Define $\boldsymbol{\phi}_t^0 = \emptyset$, the empty set, and $\mathcal{G}_0(\mathbf{x}_t; \boldsymbol{\phi}_t^0) = 1 \ \forall \mathbf{x}_t$.
   – For $k = 1, \ldots, K$
      • For $i = 1, \ldots, N$, sample $\widetilde{\mathbf{x}}_t^{(i)} \sim \pi_k(\mathbf{x}_t | \mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t}^k, \boldsymbol{\phi}_t^{k-1})$.
      • Estimate the parameters $\boldsymbol{\phi}_t^k$ of the model $\mathcal{G}_k(\mathbf{x}_t; \boldsymbol{\phi}_t^k)$, designed to
        approximate $\prod_{j=1}^k p(\mathbf{y}_t^j | \mathbf{x}_t)$, using data pairs from the training set
        $\{ \left( \widetilde{\mathbf{x}}_t^{(i)}, \mathcal{G}_{k-1}(\widetilde{\mathbf{x}}_t^{(i)}; \boldsymbol{\phi}_t^{k-1}) p(\mathbf{y}_t^k | \widetilde{\mathbf{x}}_t^{(i)}) \right); i = 1, \ldots, N \}$.
      • If $k < K$, quantize $\boldsymbol{\phi}_t^k$ and send to node $k + 1$.

3. Backward parameter exchange and weight calculation
   – For $k = K, K - 1, \ldots, 1$
      • For $i = 1, \ldots, N$, sample $\widetilde{\mathbf{x}}_t^{(i)} \sim \pi(\mathbf{x}_t | \mathbf{x}_{0:t-1}^{(i)}, \boldsymbol{\phi}_t^K)$ and set $\widetilde{\mathbf{x}}_{0:t}^{(i)} =$
        $\left( \mathbf{x}_{0:t-1}^{(i)}, \widetilde{\mathbf{x}}_t^{(i)} \right)$.
      • For $i = 1, \ldots, N$, evaluate the (approximate) importance weights

            $\widetilde{w}_t^{(i)} = \frac{\mathcal{G}_K(\widetilde{\mathbf{x}}_t^{(i)}; \boldsymbol{\phi}_t^K) \ p(\widetilde{\mathbf{x}}_t^{(i)} | \mathbf{x}_{t-1}^{(i)})}{\pi(\widetilde{\mathbf{x}}_t^{(i)} | \mathbf{x}_{0:t-1}^{(i)}, \boldsymbol{\phi}_t^K)}.$

      • Normalize the importance weights.
      • If $k > 1$, transmit the quantized parameters $\boldsymbol{\phi}_t^K$ to node $k - 1$.

4. Selection step
   – For $k = 1, \ldots, K$, resample with replacement $N$ particles $\{\mathbf{x}_{0:t}^{(i)}; i =$
     $1, \ldots, N\}$ from the set $\{\widetilde{\mathbf{x}}_{0:t}^{(i)}; i = 1, \ldots, N\}$ according to the importance
     weights.
   – Set $t \leftarrow t + 1$ and go to step 2.
```

Fig. 1. High-level algorithm description of the distributed particle filter that uses parametric models to approximate likelihood factors (see Section 3.2).

estimated using an algorithm that is not extremely computationally demanding. The sensor nodes must run a training algorithm every time-step, so the training must be a reasonable exercise given processing power and memory constraints. It should also be noted that each sensor must be aware of the functional structure of the models of its neighbours and also that of the global model $\mathcal{G}_K$.

3.3 Distributed Particle Filter using Adaptive Encoding

In this section, we describe a distributed particle filtering algorithm that uses the predictive capabilities of the particle filter located at each sensor node to perform efficient, adaptive encoding of the measured data. In contrast to the parametric algorithm, which is effective when the data dimension is high, sensor computational limitations combine with the structure of the encoding-based particle filter to impose a restriction on the feasible data dimension.

In the description of the algorithm that follows, we again consider the scenario where there is a single communication chain from node 1 to node K, with any node k in the interior of the chain communicating only with nodes $k-1$ and $k+1$. As in the parametric-approximation filter, the adaptive-encoding particle filter is equally applicable if there is a tree structure for communication.

Sensor k records a vector of measurements $\mathbf{y}_t^k$ at each time step. Denote the dimension of this vector d_k. The vector of data collected at all sensors, $\mathbf{y}_t$, has dimension $\mathbf{d} \triangleq \sum_{k=1}^K d_k$. The particle filters at all nodes are synchronized with one another, and each particle filter is initialized as in the standard sequential Monte Carlo framework, by sampling $\{x_0^{(i)}; i = 1, \dots, N\}$ from $p(\mathbf{x}_0)$. Each node also maintains a set of $\mathbf{d}$ time-varying, scalar quantizers $\mathbf{q}_t$, consisting of a codebook and a partition function. We will index the set using a pair of integers (k, j), so that at time t, $q_t^{k,j}$ represents the quantizer corresponding to the $j-th$ element of the measurement vector $\mathbf{y}_t^k$.

At time t, sensor node k samples from the prior distribution $p(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)})$ to generate N particles $\{\widetilde{\mathbf{x}}_t^{(i)}; i = 1, \dots, N\}$. (As an alternative to these steps, one can follow the idea of auxiliary variable-based particles filters [10], and have the sensor generate a set of values $\{\boldsymbol{\mu}_t^{(i)}; i = 1, \dots, N\}$, where $\boldsymbol{\mu}_t^{(i)}$ is the mean, mode, or some other likely value associated with $p(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(i)})$). Based on the set of particles $\widetilde{\mathbf{x}}_t^{(i)}$ (or the $\boldsymbol{\mu}_t^{(i)}$) , the sensor node then generates a set of predictive data values $\{\widetilde{\mathbf{y}}_t^{(i)}; i = 1, \dots, N\}$ by drawing from the likelihood functions $p(\mathbf{y}_t|\widetilde{\mathbf{x}}_t^{(i)}$ (or $p(\mathbf{y}_t|\widetilde{\boldsymbol{\mu}}_t^{(i)})$. Note that this step requires that each sensor have knowledge of the global likelihood function. The particle filters are synchronized, so every node generates the same set of predictive data values.

The next step of the algorithm is the quantizer training phase. In designing and training the quantizers $\mathbf{q}_t$, a natuaral choice is to attempt to minimize the error function $\int p(\mathbf{y}_t|\mathbf{x}_{0:t-1}) \left[\int \left(p(\mathbf{y}_t|\mathbf{x}_t) - p(\mathbf{q}_t(\mathbf{y}_t)|\mathbf{x}_t)\right)^2 d\mathbf{x}_t\right] d\mathbf{y}_t$, where $\mathbf{q}_t(\mathbf{y}_t)$ is the quantized value of $\mathbf{y}_t$. The particle approximation to this error function is $\sum_{i=1}^N \sum_{j=1}^N \left(p(\widetilde{\mathbf{y}}_t^{(i)}|\widetilde{\mathbf{x}}_t^{(j)}) - p(\mathbf{q}_t(\widetilde{\mathbf{y}}_t^{(i)})|\widetilde{\mathbf{x}}_t^{(j)})\right)^2$. Vector quantization can be explored as a possible path for minimizing this function, but in any event, sensor

Distributed Adaptive Encoding-based Particle Filter

1. *Initialisation*, $t = 0$.
   - Synchronize the particle filters by equating the random seeds.
   - For each sensor $k = 1, \ldots, K$
     - For $i = 1, \ldots, N$, sample $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$ and set $t = 1$.
2. *Quantizer training step*
   - For each sensor $k = 1, \ldots, K$
     - For $i = 1, \ldots, N$, sample $\widetilde{\mathbf{x}}_t^{(i)} \sim p(\mathbf{x}_t | \mathbf{x}_{0:t-1}^{(i)})$ and $\widetilde{\mathbf{y}}_t^{(i)} \sim p(\mathbf{y}_t | \widetilde{\mathbf{x}}_t^{(i)})$.
     - For each element $y_t^{k,j}$ of the data $\mathbf{y}_t$, train a $D_{k,j}$ element quantizer $q_t^{k,j}$ using the Lloyd-Max algorithm applied to the data set $\{\widetilde{\mathbf{y}}_t^{(i)}; i = 1, \ldots, N\}$ using the error function

$$\sum_{i=1}^{N} \left(p(\widetilde{\mathbf{y}}_t^{(i)} | \widetilde{\mathbf{x}}_t^{(i)}) - p\left(\widetilde{\mathbf{y}}_t^{\overline{k,j},(i)}, q_t^{k,j}(\widetilde{y}_t^{k,j,(i)}) \, | \widetilde{\mathbf{x}}_t^{(i)} \right) \right)^2 . \qquad (1)$$

3. *Quantization*
   - For each sensor k, quantize $\mathbf{y}_t^k$ using the set of quantizers $\mathbf{q}_t^k = \{q_t^{k,j}\}$ to form quantized data $\left(\mathbf{y}_t^k\right)'$.
   - Distribute the quantized data $\mathbf{y}_t' = \{\left(\mathbf{y}_t^k\right)'; k = 1, \ldots, K\}$ to all sensors.
4. *Importance sampling*
   - For each sensor $k = 1, \ldots, K$
     - For $i = 1, \ldots, N$, sample $\widetilde{\mathbf{x}}_t^{(i)} \sim \pi(\mathbf{x}_t | \mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t}')$, and set $\widetilde{\mathbf{x}}_{0:t}^{(i)} = \left(\mathbf{x}_{0:t-1}^{(i)}, \widetilde{\mathbf{x}}_t^{(i)}\right)$.
     - For $i = 1, \ldots, N$, evaluate the (approximate) importance weights

$$\widetilde{w}_t^{(i)} = \frac{p(\mathbf{y}_t' | \widetilde{\mathbf{x}}_t^{(i)}) \; p(\widetilde{\mathbf{x}}_t^{(i)} | \mathbf{x}_{t-1}^{(i)})}{\pi(\widetilde{\mathbf{x}}_t^{(i)} | \mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t}')} .$$

     - Normalize the importance weights.
5. *Selection step*
   - For $k = 1, \ldots, K$, resample with replacement N particles $\{\mathbf{x}_{0:t}^{(i)}; i = 1, \ldots, N\}$ from the set $\{\widetilde{\mathbf{x}}_{0:t}^{(i)}; i = 1, \ldots, N\}$ according to the importance weights.
   - Set $t \leftarrow t + 1$ and go to step 2.

Fig. 2. High-level algorithm description of the distributed particle filter that performs adaptive encoding of sensor data (see Section 3.3).

k does not have access to $\mathbf{y}_t$. Sensor k requires a quantizer $\mathbf{q}_t^k$ that maps $\mathbf{y}_t^k$ to a quantized value. In the algorithm proposed here, we take this one step further, and consider scalar quantizations, training $\mathbf{q}_t^{k,j}$ at each time step to quantize individual measurements $y_t^{k,j}$. In training $q_t^{k,j}$, we use the error function: $\int p(\mathbf{y}_t|\mathbf{x}_{0:t-1}) \left[\int \left(p(\mathbf{y}_t|\mathbf{x}_t) - p(\mathbf{y}_t^{\overline{k,j}}, q_t^{k,j}(y_t^{k,j})|\mathbf{x}_t) \right)^2 d\mathbf{x}_t \right] d\mathbf{y}_t,$ where $\mathbf{y}_t^{\overline{k,j}}$ denotes all elements of by_t except $y_t^{j,k}$. The particle approximation to this error function is:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \left(p(\widetilde{\mathbf{y}}_t^{(i)}|\widetilde{\mathbf{x}}_t^{(j)}) - p(\widetilde{\mathbf{y}}_t^{\overline{k,j},(i)}, q_t^{k,j}(\widetilde{y}_t^{k,j,(i)})|\widetilde{\mathbf{x}}_t^{(j)}) \right)^2 . \tag{2}$$

In the simulations in this paper we have used the Lloyd-Max approach to train the scalar quantizers [11, 12] (it is also possible to consider computationally simpler methods or entropy-coded quantization approaches [13]). Note that using the error function (2) involves the evaluation of N^2 likelihoods upon every iteration of the algorithm. We have observed in simulations that reducing this to N evaluations by only including the terms $i = j$ results in similar filter performance. Our goal is to determine a good quantizer (not necessarily the optimal quantizer), so we can also limit the number of training iterations. However, even with these modifications, the training of several Lloyd-Max quantizer is a computationally demanding exercise. This results in limitations on the viable data dimension per time step for this filtering approach.

Once the quantizers have been trained at all sensor nodes, the sensor data are quantized to $\mathbf{y}'_t \triangleq \mathbf{q}_t(\mathbf{y}_t)$ and distributed throughout the network. The sensors then perform the standard particle filtering steps (importance sampling, weight evaluation, and particle selection) based on the quantized data $\mathbf{y}'_t$. Figure 2 presents the algorithm in high-level pseudo-code format.

The only reduction in communication cost compared to the dissemination of the raw data lies in the achievable compression in the quantization. The communication cost in bits per time step, neglecting overhead bits, is $\sum_{k=1}^{K} D_k M_k$, where D_k is the number of bits required after quantization and M_k is the number of communication hops required to send sensor k's quantized data throughout the sensor network. As illustrated in Section 5, substantial compression can be achieved whilst maintaining estimation accuracy. The algorithm's computational complexity grows linearly in data dimension (the quantizer training is the dominant computational expense), and this restricts the feasible size of the sensor network for real-time operation.

4 A Hierarchical Sensor Network

For the remainder of the paper, we focus on the distributed particle filtering algorithm based on adaptive data encoding. It is clear from the preceding section that the algorithm becomes impractical for a large number of sensors. Every sensor must maintain and train a codebook for each scalar measurement in the network, and the computational expense incurred in this exercise soon becomes overwhelming. Moreover, every sensor must stay awake and participate in the algorithm every time step, so that filters do not lose synchronization. In this section, we describe the high-level structure of a hierarchical sensor network that addresses these issues.

In the hierarchical network, we consider that there are two classes of sensor nodes, which we denote classes A and B for convenience. Class A nodes have substantially more energy and computational power than the more numerous class B nodes. Class B nodes are responsible for sensing the monitored environment and reporting their measurements to a single parent class A node. We assume that the density of class A nodes is sufficient that each class B node can directly communicate with at least one class A node. Class A nodes are responsible for performing all computation and managing the class B sensor nodes. Class A nodes are always active; a class B node is activated for measurement and communication by its parent node.

Due to the limitations of the adaptive encoding particle filtering algorithm, the class A nodes, numbered $1, \ldots, K$ only activate a small number of class B nodes (for measurement) per time step. We denote this set of sensors by $\mathcal{V}_t$. Other class B nodes may be active for the purpose of relaying messages between the set of class A nodes.

The class A nodes implement the distributed particle filtering algorithm described in Section 3.3. At time $t = 0$, the algorithm is initialized by sampling particles at each class A node from $p(\mathbf{x}_0)$, and a random set of sensors $\mathcal{V}_1$ is selected for the first measurement. Each sensor $v \in \mathcal{V}_1$ makes its set of measurements $\mathbf{y}_1^v$ and transmits the data to its parent class A node. The transmission at this step involves fine quantization (of the order of 16 or 32 bits per measurement). The K class A nodes then perform the distributed particle filtering algorithm exactly as described in Figure 2. The nodes only generate a small number of linear quantizers, one for each data measurement made by the active class B sensor nodes, so the computational requirements are manageable. The exchange between the class A nodes involves highly compressed data (2-5 bits per measurement).

At a subsequent time step t, instead of a random set of sensors being activated for measurement, the class A nodes decide upon a set based on a prediction of which sensors will provide the most informative measurements. Algorithms for performing this type of sensor management exercise have been described in [1,9, 14]. As the decision is made on the basis of a common particle filter, every class A node is aware of the set of active sensor nodes that will perform measurement one step ahead in time. This is important because it means that the number of bits required for data labelling is minimal, and related to the cardinality of the active set rather than the total number of sensors. The number of information bits that need to be transferred throughout the network is $\sum_{k=1}^{K} D_k$, where D_k is the number of bits required by class A node k. This expression excludes the initial finely-quantized communications between the active class B sensors and their parents.

5 A Simulation Example: Tracking a Manoeuvring Object

In this section, we investigate an example of the application of the adaptive encoding distributed particle filter using the hierarchical sensor network framework described in Section 4. We consider the exercise of tracking an object manoeuvring in a 2-D plane. The dynamic system (a jump-state Markov model) is described by an initial distribution $p(u_0, \theta_0, \mathbf{x}_0)$ and the update equations

$$u_t \sim p(u_t | u_{t-1}), \tag{3}$$

$$\theta_t = \theta_{t-1} + c(u_t) + v_t, \tag{4}$$

$$\mathbf{x}_t = \mathbf{x}_{t-1} + m[\cos\theta_t,\, \sin\theta_t]. \tag{5}$$

Here $u_t \in 0, 1, 2$ is a time varying state, indicating no turn ($c(0) = 0$), left-turn ($c(1) = 0.1$ radians), and right-turn ($c(2) = -0.1$ radians), respectively. The angle of motion is determined by θ_t, v_t is the innovation noise (Gaussian), and x_t is the position. The velocity is constant and specified by m.

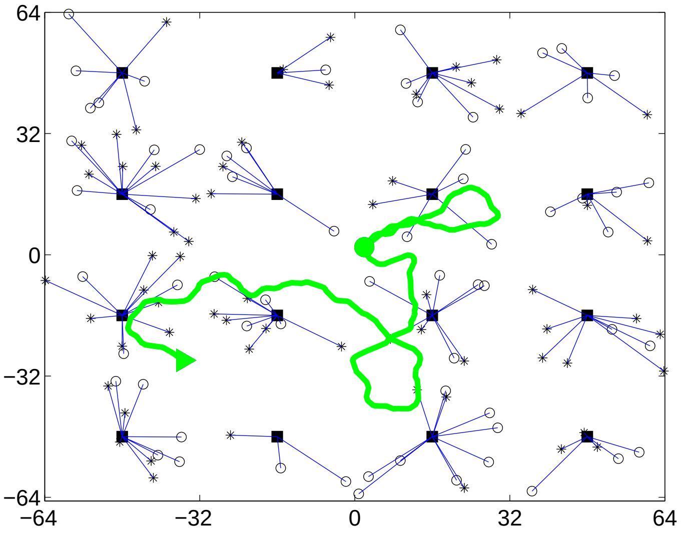

Fig. 3. An example realization of the sensor network for the simulation in Section 5. Solid squares indicate class A nodes at fixed locations, equally spaced in the plane. Circles are class B angle-measurement nodes; stars are class B distance-measurement nodes. Class B nodes are uniformally distributed. The thin lines indicate the class A parent of each class B node. The thick line indicates an example trajectory of the object over 500 time steps, starting at the solid circle and moving to the triangle.

In our system, there are two types of class B nodes: those capable of measuring the angle of the object's position relative to the node ϕ_t, and those capable of measuring distance to the object r_t. The observation equations for a node v with position g_v are:

$$\phi_t^v = \arctan(x_t - g_v) + n_t \tag{6}$$

$$r_t^v = \max(||x_t - g_v|| + s_t, 0) \tag{7}$$

The noise terms, n_t and s_t are modelled as zero-mean Gaussian with variances σ_n^2 and σ_s^2, respectively.

We performed simulations with 128 class B sensors, 64 of each type, and 16 class A nodes. The positions of the class B nodes in the plane (covering $[-64, 64] \times [-64, 64]$) were random, drawn according to a uniform distribution. The class A nodes were equally spaced across the plane. Figure 3 depicts an example realization of the sensor field.

In our studies, the object was tracked for 500 time steps. The object's initial position was determined by a Gaussian distribution centred at [2,2] and diagonal covariance entries set to 1. The initial angle of motion of the object was determined by a Gaussian, centred at $\pi/4$, with variance 0.01, and u_0 was set to 0. The object moved with a constant velocity $m = 0.5$, and the innovation noise had variance 0.001. The observation noise in our simulations was generated by setting $\sigma_n = 0.02$ and $\sigma_s = 0.02$. The state transition probability matrix $p(u_t|u_{t-1})$ was set as:

$$\begin{pmatrix} 0.75 & 0.65 & 0.65 \\ 0.125 & 0.3 & 0.05 \\ 0.125 & 0.05 & 0.03 \end{pmatrix}$$

At any time step, the class A nodes activate eight class B nodes for measurement, including four angle-measurement sensors and four distance-measurement sensors. These are chosen as the closest (of their kind) to the predicted position of the object, the prediction being made one step-ahead.

We conducted a Monte Carlo study of the performance of the adaptive encoding distributed particle filter. The study involved $S = 20$ realizations of the sensor field and the object path. For each study, the particle filter was applied with $REP = 10$ different random seeds for initialization. In our simulations, the particle filters knew the model parameters, and N, the number of particles per filter, was set to 500. We examined the performance for three fixed settings of D_v, the number of bits used for data representation by each active class B sensor (and per measurement in this case). We compared the estimation performance of $D_v = \{2, 3, 4\}$ to the raw data case, where we use 16 bits to represent each measurement via straightforward quantization over the feasible range.

For each setting of D_v, we calculated the mean-squared error between the true object position and the particle filter-based mean-square estimate of object-position. This was derived for each realization s by averaging over the REP instances of the algorithm. Based on this we obtained a log mean squared-error measure:

$$LMSE_t = \log \frac{1}{S} \sum_{s=1}^{S} \left[\frac{1}{REP} \sum_{r=1}^{REP} ||\mathbf{x}_{t,s} - \widehat{\mathbf{x}}_{t,r,s}||^2 \right] \tag{8}$$

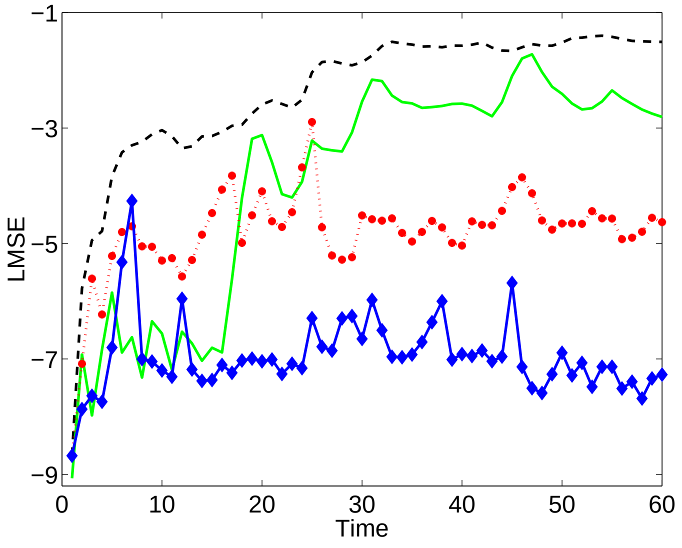

Fig. 4. Plot of mean squared error performances of the adaptive encoding distributed particle filter at various quantization levels. Dashed line corresponds to a 2-bit quantization; solid line to a 3-bit quantization; dotted line with * markers to a 4-bit quantization; solid line with diamond markers to a non-adaptive 16-bit quantization.

Figure 4 plots the LMSE for the four quantization levels over the first 60 time instants. The algorithm performance is as expected, with estimation performance improving (exponentially) as the number of bits increases. The loss in accuracy through the use of four-bit encoding as opposed to non-adaptive 16-bit quantization of the raw data is not particularly substantial, particularly considering the reduction in communication cost. The total number of information bits transmitted by the sensor network is $128 + 8D_vM$, where M is the number of hops needed to send a measurement to all class A nodes. In moving from the non-adaptive 16-bit to the adaptive 4-bit quantization, we achieve (approximately) a four-fold saving in communication.

6 Discussion and Conclusions

We have presented two distributed particle filtering algorithms for tracking posterior distributions in Markovian state-space models using sensor networks. The first algorithm is applicable to state-space models for which it is possible to factorize the likelihood function and approximate the factors using low-dimensional parametric models. It results in a substantial communication saving in situations where the data dimension per time step is large. The second algorithm uses distributed particle filters to adapt linear quantizers for individual sensor measurements. Whilst the latter algorithm is applicable to more general models, it cannot be applied when the data dimension is high, because substantial computation is needed to train efficient quantizers. In light of this, we described a hierarchical sensor network that supports implementation of the algorithm.

Both algorithms have several limitations and the treatment in this paper has glossed over some issues. The algorithms require that each sensor has a knowledge of the global likelihood function. Neither algorithm provides a mechanism for handling local state parameters (such as sensor position) separately from the global state. For the parametric approximation-based algorithm, methods for identifying and training appropriate parametric models for the likelihood factors remain to be developed. In future research, we hope to provide a more refined characterization of the state-space models for which the parametric approach is applicable. In the case of the adaptive encoding algorithm, the restriction to a small data dimension is frustrating. We are currently exploring data aggregation and vector quantization strategies in our attempts to alleviate the restriction.

References

1. Liu, J., Liu, J., Reich, J., Cheung, P., Zhao, F.: Distributed group management for track initiation and maintenance in target localization applications. In: Proc. IEEE Conf. Information Processing in Sensor Networks, Palo Alto, CA (2003)
2. Mutambara, A.: Decentralized estimation and control for multisensor systems. CRC Press, Boca Raton, FL (1998)
3. Kam, M., Zhu, X., Kalata, P.: Sensor fusion for mobile robot navigation. Proc. IEEE **85** (1997) 108–119
4. Rosencrantz, M., Gordon, G., Thrun, S.: Decentralized sensor fusion with distributed particle filters. In: Proc. Conf. Uncertainty in Artificial Intelligence, Acapulco, Mexico (2003)
5. Anderson, B., Moore, J.: Optimal Filtering. Prentice-Hall, New Jersey (1979)
6. Bucy, R., Senne, K.: Digital synthesis of nonlinear filters. Automatica **7** (1971) 287–298
7. Doucet, A., de Freitas, N., Gordon, N., eds.: Sequential Monte Carlo Methods in Practice. Series: Statistics for Engineering and Information Science. Springer-Verlag, New York (2001)
8. Zhao, F., Shin, J., Reich, J.: Information-driven dynamic sensor collaboration for tracking applications. IEEE Signal Processing Magazine **19** (2002) 61–72
9. Shin, J., Guibas, L., Zhao, F.: A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In: Proc. IEEE Conf. Information Processing in Sensor Networks, Palo Alto, CA (2003)
10. Pitt, M., Shephard, N.: Filtering via simulation: auxiliary particle filters. J. Amer. Stat. Assoc. **94** (1999) 590–599
11. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Info. Theory **28** (1982) 129–137
12. Max, J.: Quantizing for minimum distortion. IRE Trans. Info. Theory **6** (1960) 7–12
13. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer Academic Press, Boston MA (1992)
14. Kreucher, C., Castella, K., Hero, A.O.: Multitarget sensor management using alpha divergence measures. In: Proc. IEEE Conf. Information Processing in Sensor Networks, Palo Alto, CA (2003)