Signal Processing 98 (2014) 121–134

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

A distributed particle filter for nonlinear tracking in wireless sensor networks

Jesse Read*, Katrin Achutegui, Joaquín Míguez

Avenida de la Universidad 30, 28911 Leganés (Madrid), Spain

ARTICLE INFO

Article history:
Received 15 May 2013
Received in revised form
28 October 2013
Accepted 16 November 2013
Available online 23 November 2013

Keywords:
Wireless sensor network
Particle filters
Distributed filtering
Target tracking

ABSTRACT

The use of distributed particle filters for tracking in sensor networks has become popular in recent years. The distributed particle filters proposed in the literature up to now are only approximations of the centralized particle filter or, if they are a proper distributed version of the particle filter, their implementation in a wireless sensor network demands a prohibitive communication capability. In this work, we propose a mathematically sound distributed particle filter for tracking in a real-world indoor wireless sensor network composed of low-power nodes. We provide formal and general descriptions of our methodology and then present the results of both real-world experiments and/or computer simulations that use models fitted with real data. With the same number of particles as a centralized filter, the distributed algorithm is over four times faster, yet our simulations show that, even assuming the same processing speed, the accuracy of the centralized and distributed algorithms is practically identical. The main limitation of the proposed scheme is the need to make all the sensor observations available to every processing node. Therefore, it is better suited to broadcast networks or multihop networks where the volume of generated data is kept low, e.g., by an adequate local pre-processing of the observations.

1. Introduction

Due to the falling size and costs of suitable hardware, the deployment of wireless sensor networks (WSNs) is becoming an increasingly attractive option for a growing number of tracking applications. Examples include security and surveillance [5], environmental monitoring (tracking of weather patterns and pollutants) [32], monitoring in domestic situations (such as in care for the elderly) [22], and biology (tracking of populations or individual animals) [27].

Distributed applications of tracking are particularly interesting in situations where high powered centralized hardware cannot be used. For example, in deployments where computational infrastructure and power are not available or where there is no time or trivial way of connecting to it. In these scenarios, and possibly many others, the signal processing tasks associated to target tracking need to be shared by the multiple nodes of the WSN. Note that many items in the literature often refer to WSNs as being "distributed", even when processing is centralized, because they are merely referring to the physically distributed nature of WSNs. See, for example [9,18]. In this paper we refer to "distributed" specifically with regard to processing, meaning that the computational tasks are divided among a set of low-power devices in the WSN.

1.1. Distributed particle filters

Stochastic filtering methods [3] are obvious candidates for tracking applications and so they have been researched by many authors in the context of WSNs [29,14,11]. Such work includes, e.g., networks of interacting Kalman filters [30], although in this case the emphasis is on the minimization of the communications among nodes, rather than the sharing of the computational load.

* Corresponding author.
E-mail address: jesse@tsc.uc3m.es (J. Read).

0165-1684/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sigpro.2013.11.020

Particle filtering for target tracking in WSNs has already attracted some attention (see, for example, [5,1,2]), including a body of work in distributed methods [10,21,7]. Its relation with agent networks has also been explored in [20]. In [7], a fully decentralized particle filtering algorithm for cooperative blind equalization is introduced. The technique is proper, in the sense that it does not make any approximations in the computation of the importance weights of the particles. However, the scheme is applicable only when the state signal is discrete, and would be infeasible in terms of computation and communication among nodes (the authors provide a simulation only) in WSNs such as we consider. In [10], the communication load is reduced using quantization and parametric approximations of densities. A similar parametric approach is applied in [21] to further simplify communications. Even though these parametric approximations are practical for their implementation on a WSN they compromise the estimation accuracy and, to our view, take away the main advantage of the particle filter (PF): its generality and capability to perform numerical inference, with full theoretical guarantees, on arbitrary state-space dynamical systems.

Recently, a class of interacting PFs has been proposed for multi-target tracking [13]. This class of algorithms relies on splitting the state-space into lower dimensional subspaces in order to become computationally tractable, but does not guarantee that the particles are assigned proper weights.

The majority of existing contributions related to particle filtering and WSNs, only offer a theoretical perspective and/or computer simulation studies, owing in part to the challenges of real-world deployment and testing on low-powered hardware. Deployments of physical sensor networks have so far been almost exclusively centralized implementations (from the computational point of view held in this paper). For instance, in [2], 25 acoustic sensors are used with a centralized PF to track a remote-controlled car; the authors of [1] use the received signal strength (RSS) measurements to track an additional moving target node. There are a few exceptions of actually distributed PFs, but they are approximations to the centralized PF whose convergence cannot be guaranteed [26].

The use of the non-parametric loopy belief propagation (NPBP) algorithm has also been suggested for localization and tracking [33] in the same context as particle filtering. This algorithm is an extension of the belief propagation algorithm to continuous variables and it uses a graph to represent the decomposition of the joint posterior. Its main advantage resides in exploiting the rich underlying structure that often arises in image processing or in sensor self-localization. Unfortunately, the NPBP scheme poses difficulties for a real-time implementation and there is, to the best of our knowledge, no rigorous proof of convergence for this algorithm.

1.2. Distributed resampling with non-proportional allocation (DRNA)

Particle filtering algorithms involve three basic steps: generation of samples, computation of weights and resampling [12]. While it is straightforward to parallelize the first two steps, resampling requires the joint processing of all the samples in the filter and so becomes a computational bottleneck. The distributed resampling with non-proportional allocation (DRNA) algorithm [6] (see also [28] for some further analysis) addresses the parallelization of the resampling step to remove this bottleneck.

The DRNA algorithm was devised to speed up the computations in particle filtering. The basic assumption in [6] is the availability of a set of processors interconnected by a high-speed network, in the manner of state-of-the-art graphical processing unit (GPU) based systems [25]. Such network is intended to guarantee that all processors in the system have access to the full set of observed data.

In a typical WSN, the communications among nodes are subject to various constraints (i.e., transmission capacity, power consumption or error rates), hence the hardware setup is fundamentally different from the one assumed in [6] or [28]. The issue of whether the DRNA scheme can be efficiently exploited in a typical WSN deployment has not been addressed, to the best of our knowledge, neither experimentally nor even by realistic simulation studies.

1.3. Contribution and organization

In this work we tackle the problem of implementing the DRNA algorithm in a practical WSN. We first revisit the standard PF and its combination with the DRNA algorithm, providing a formal description of the methodology. This includes simple analysis showing that (a) the importance weights are proper and (b) the resampling scheme is unbiased. While intuitively expected, these two simple results had not been given explicitly in [6,28].

In the second part of the paper, we address the practical implementation of a distributed PF for target tracking, based on the DRNA scheme, that runs in real time over a WSN. We have developed a software and hardware testbed implementing the required algorithmic and communication modules in physical nodes, equipped with light-intensity sensors but with limited processing and communication capabilities. We assess the tracking performance of the resulting system in terms of the tracking error obtained with both synthetic and real data. Finally, we study the constraints in the real-time operation and the communication capabilities (compared to a centralized PF) by way of experiments with our testbed implementation.

The main limitation of the proposed scheme is that every node performing a subset of the computations of the PF should have access to all the observations (i.e., all the measurements collected by the WSN at the current time step) in order to guarantee that the particle weights are proper and, therefore, the resulting estimators consistent. The DRNA-based PF, therefore, is better suited to broadcast networks or multihop WSNs where the volume of data generated per time step is limited. The algorithm can still be applied with different sets of observations available at each node, but in that case the particle weights are only proper locally (at each node) and not globally (over the whole network).

The rest of the paper is organized as follows. In Section 2 we introduce the problem of target tracking in the context of

Bayesian filtering and describe the solution to the non-linear filtering problem with a centralized PF. In Section 3 we provide a formal description on the DRNA algorithm. Section 4 describes the software and hardware framework for the testbed implementation of the distributed tracker. In Section 5 we provide details of the specific deployment for the experiments. Both numerical and experimental results are presented and discussed in Section 6 and, finally, Section 7 is devoted to the conclusions.

2. Non-linear filtering in state-space systems

2.1. Bayesian filtering

Consider the Markov state-space random model with conditionally independent observations [16, chapter 1], [31] described by the triplet[1]

$$p(\mathbf{x}_0), \quad p(\mathbf{x}_t|\mathbf{x}_{t-1}), \quad p(\mathbf{y}_t|\mathbf{x}_t), \quad t=1,2,\ldots \tag{1}$$

where $\mathbf{x}_t$ is a $d_x \times 1$ vector containing the state variables at time $t \geq 0$, $\mathbf{y}_t$ is a $d_y \times 1$ vector of observations at time $t \geq 1$, $p(\mathbf{x}_0)$ is the prior probability density function (pdf) of the state, the transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ describes the (random) dynamics of the process $\mathbf{x}_t$ and the conditional pdf $p(\mathbf{y}_t|\mathbf{x}_t)$ describes how the observations are related to the state and it is usually referred to as the likelihood of $\mathbf{x}_t$. The goal of a stochastic filtering algorithm is to approximate the sequence of posterior pdfs $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, $t \geq 1$. In particular, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is often termed the filter density at time t.

Assume that the pdf $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$ is available. The filter density at time t can be recursively computed in two steps. We first compute the predictive pdf

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, d\mathbf{x}_{t-1} \tag{2}$$

and, when the new observation $\mathbf{y}_t$ is collected, we update $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ to obtain

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) \tag{3}$$

Eqs. (2) and (3) form the basis of the optimal Bayesian solution to the filtering problem. Knowledge of the posterior density $p(\mathbf{x}_t|\mathbf{y}_t)$ enables the computation of optimal estimators of the state according to different criteria. For example, the minimum mean-square error (MMSE) estimator is the posterior mean of $\mathbf{x}_t$, i.e., $\mathbf{x}_t^{MMSE} = \int \mathbf{x}_t\, p(\mathbf{x}_t|\mathbf{y}_{1:t})\, d\mathbf{x}_t$.

If the system of Eq. (1) is linear and Gaussian, then $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is Gaussian and can be obtained exactly using the Kalman filter algorithm [23]. If the state space is discrete and finite, exact solutions can also be computed [31]. However if any of the pdf's in (1) is non-Gaussian, or the system is nonlinear, we have to resort to suboptimal algorithms in order to approximate the filter pdf $p(\mathbf{x}_t|\mathbf{y}_{1:t})$.

[1] We use letter p to represent both probability density functions (pdfs) and probability mass functions (pmfs). This is an argument-wise notation: if x and y are continuous random variables (r.v.), then $p(x)$ is the pdf of x and $p(y)$ is the pdf of y, possibly different. If z is a discrete r.v., $p(z)$ is the pmf of z. Conditional pdfs and pmfs are indicated in the obvious manner, e.g., $p(x|y)$ is the conditional pdf of the r.v. x given the r.v. y. This notation is common in Bayesian analysis and in the particle filtering literature (see, e.g., [17,16,12]).

2.2. Particle filtering

PFs, also known as sequential Monte Carlo methods, are simulation based algorithms that yield estimates of the state based on a random point-mass (or "particle") representation of the probability measure with density $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ [16,3]. It is often convenient to derive PFs as instances of the SIS methodology [17]. Consider the joint posterior density $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$. Bayes' theorem, together with the Markov property of the state and the conditional independence of the observations, yields the recursive relationship

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1}), \quad t \geq 1, \tag{4}$$

where we have omitted the proportionality constant $1/p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$, which is independent of $\mathbf{x}_{0:t}$.

Most PFs can be obtained as a combination of (4) with the importance sampling (IS) principle [16, chapter 1]. In particular, if we draw a collection of sample sequences $\mathbf{x}_{0:t}^{(m)}$, $m=1,\ldots,M$, from a proposal density $\pi_t(\mathbf{x}_{0:t})$ that admits a factorization $\pi_t(\mathbf{x}_{0:t}) = \pi_{t|t-1}(\mathbf{x}_t|\mathbf{x}_{0:t-1})\pi_{t-1}(\mathbf{x}_{0:t-1})$, then the samples are assigned nonnormalized importance weights of the form

$$\begin{aligned} w_t^{(m)*} &= \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(m)}) p(\mathbf{x}_t^{(m)}|\mathbf{x}_{t-1}^{(m)}) p(\mathbf{x}_{0:t-1}^{(m)}|\mathbf{y}_{1:t-1})}{\pi_{t|t-1}(\mathbf{x}_t^{(m)}|\mathbf{x}_{0:t-1}^{(m)})\pi_{t-1}(\mathbf{x}_{0:t-1}^{(m)})} \\ &\propto \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(m)}) p(\mathbf{x}_t^{(m)}|\mathbf{x}_{t-1}^{(m)})}{\pi_{t|t-1}(\mathbf{x}_t^{(m)}|\mathbf{x}_{0:t-1}^{(m)})} w_{t-1}^{(m)}, \end{aligned}$$

where $w_{t-1}^{(m)}$ is the normalized weight of $\mathbf{x}_{0:t-1}^{(m)}$. Normalization is straightforward, we simply compute $w_t^{(m)} = w_t^{(m)*} / \sum_{j=1}^{M} w_t^{(j)*}$. Let us also note that drawing from $\pi_t(\mathbf{x}_{0:t})$ can be done sequentially: given $\mathbf{x}_{0:t-1}^{(m)}$, we simply generate $\mathbf{x}_t^{(m)}$ from the pdf $\pi_{t|t-1}(\mathbf{x}_t|\mathbf{x}_{0:t-1}^{(m)})$.

The simplest implementation of the PF is obtained when we choose $\pi_t(\mathbf{x}_{0:t}) = p(\mathbf{x}_{0:t}) = p(\mathbf{x}_0)\prod_{k=1}^{t} p(\mathbf{x}_k|\mathbf{x}_{k-1})$ and perform resampling steps [17] at every time step. The resulting algorithm is often called bootstrap filter [19] or sequential importance resampling (SIR) algorithm [17]. We outline it in Table 1 and will henceforth refer to it as a centralized particle filter (CPF).

The resampling step randomly eliminates samples with low importance weights and replicates samples with high importance weights in order to avoid the degeneracy of the importance weights over time [17,31]. Here, we apply multinomial resampling, but several other choices exist [8,17,15,3]. We also assume that resampling is carried out at every time step, but applying the proposed methods with periodic or random resampling times is straightforward.

Bayesian estimators of $\mathbf{x}_t$ can be easily approximated using the random grid $\{\overline{\mathbf{x}}_t^{(m)}, w_t^{(m)}\}_{m=1}^{M}$. In particular, let $\mu_t(d\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{y}_{1:t})\, d\mathbf{x}_t$ be the probability measure associated to the posterior pdf $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. If we define the discrete random measure

$$\mu_t^M(d\mathbf{x}_t) = \sum_{m=1}^{M} w_t^{(m)} \delta_{\overline{\mathbf{x}}_t^{(m)}}(d\mathbf{x}_t),$$

where $\delta_{\overline{\mathbf{x}}_t^{(m)}}(d\mathbf{x}_t)$ is the unit delta measure located at $\overline{\mathbf{x}}_t^{(m)}$, then we can approximate any integrals with respect to the probability measure $\mu_t(d\mathbf{x}_t)$ as weighted sums. For

Table 1
Centralized implementation of a particle filter (CPF).

Initialization at $t=0$:
1. Draw M random samples, $\mathbf{x}_0^{(m)}$, $m=1,\ldots,M$, from prior $p(\mathbf{x}_0)$.

Recursive step, for $t>0$:
1. For $m=1,\ldots,M$,
   - draw $\overline{\mathbf{x}}_t^{(m)} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(m)})$ and set $\overline{\mathbf{x}}_{0:t}^{(m)}=\{\overline{\mathbf{x}}_t^{(m)}, \mathbf{x}_{0:t-1}^{(m)}\}$;
   - compute importance weights $w_t^{(m)*}=p(\mathbf{y}_t|\overline{\mathbf{x}}_t^{(m)})$.
2. Normalize the weights as $w_t^{(m)}=w_t^{(m)*}/\sum_{j=1}^{M} w_t^{(j)*}$.
3. Resample the weighted sample $\{\overline{\mathbf{x}}_{0:t}^{(m)}, w_t^{(m)}\}_{m=1}^{M}$ to obtain an unweighted sample $\{\mathbf{x}_{0:t}^{(m)}\}_{m=1}^{M}$.

example, the MMSE estimator of $\mathbf{x}_t$ can be approximately computed as

$$\mathbf{x}_t^{MMSE}=\int \mathbf{x}_t \mu_t(d\mathbf{x}_t) \approx \int \mathbf{x}_t \mu_t^M(d\mathbf{x}_t)=\sum_{m=1}^{M} w_t^{(m)} \overline{\mathbf{x}}_t^{(m)}.$$

We refer to the standard PF of Table 1 as centralized in order to make explicit that it requires a central unit (CU) that collects all the observations together, generates all the particles and processes them together. A number of results regarding the convergence of the CPF can be found in [3] and references therein.

3. Distributed particle filtering

3.1. General structure

We look at the DRNA algorithm introduced in [6]. This algorithm was originally proposed to speed up the processing time of PFs by making them suitable for multiprocessor devices endowed with high-speed communication networks. In this paper, we propose to apply a DRNA scheme to implement a distributed PF on a WSN, whose nodes can operate as processing elements (PEs). Let us remark that this framework is rather different from the one assumed in [6] or [28]. In particular, the PEs are low-powered devices that have to perform sensing, computation and radio communication tasks while running on batteries. Moreover, the schemes of [6,28] are based on the assumption that all observations can be readily made available to all PEs in the system. Such capacity cannot be taken for granted in a WSN, where the observations are collected locally by the nodes and communications are necessarily constrained because of energy consumption. In the following we describe the method using, essentially, the notation of [28].

Assume we have N processing nodes (i.e., N PEs) in the network; each is capable of running a separate PF with K particles (we ignore any non-processing nodes for now since they do not run particle filters). The total number of particles distributed over the network is $M=NK$. In particular, after the completion of a full recursive step of the distributed PF at time $t-1$, the n-th PE should hold the set $\{\mathbf{x}_{t-1}^{(n,k)}, w_{t-1}^{(n,k)*}, W_{t-1}^{(n)*}\}_{k=1,\ldots,K}$, where

- $\mathbf{x}_{t-1}^{(n,k)}$ is the k-th particle at the n-th PE,
- $w_{t-1}^{(n,k)*}$ is the corresponding nonnormalized importance weight, and
- $W_{t-1}^{(n)*}=\sum_{k=1}^{K} w_{t-1}^{(n,k)*}$ is the nonnormalized aggregated weight of PE n.

Each PF run locally in a node involves the usual steps of drawing samples, computing weights and resampling. In particular, resampling is carried out only locally, without interaction with the particles in other PEs. In order to avoid the degeneracy of the local sets of particles (e.g., when K is very low), the DRNA scheme includes a particle exchange step in which the PEs interchange subsets of their particles and nonnormalized weights. This step involves the update of the aggregated weights, but each individual particle preserves its nonnormalized importance weight, no matter its location.

Compared to the implementation of a DRNA-based particle filter in a single machine (as assumed in [6,28]) the design of a computational scheme for a WSN arises issues related to the exchange of data among the PEs, either particles or observations. In the particle exchange step, we need to introduce additional notation in order to define neighbor PEs, i.e., those nodes of the WSN that run a local PF and have a communication link that enables the transmission of a subset of their particles within the time frame of a single sequential step of the filter. The spread of the observations over the WSN is not dealt directly in this section, but we advocate the local processing of the measurements collected by the sensors in order to reduce the volume of transmitted data. This is the strategy followed in the real time implementation discussed in Sections 4 and 5.

In the following, we describe the algorithm steps in detail, including the computation of state estimators, and a complete outline of the method.

3.2. Particle exchange

The particle set in the n-th PE is said to degenerate when its aggregated weight $W_t^{(n)*}$ becomes negligible compared to the aggregated weights of the other nodes. Note that the value of the normalized aggregated weight is then close to zero, $W_t^{(n)}=W_t^{(n)*}/\sum_{k=1}^{N} W_t^{(k)*}\approx 0$ and the particles in the n-th set hardly contribute to the approximation of the posterior probability distribution of interest. This means that the computational effort invested in propagating them becomes a waste.

In order to keep the aggregated weights balanced, neighboring nodes can exchange subsets of particles and local nonnormalized weights [28]. Assume that, at the beginning of the t-th time step, the n-th PE holds the

weighted particles $\{\mathbf{x}_{t-1}^{(n,k)}, w_{t-1}^{(n,k)^*}\}_{k=1,\dots,K}$ (note the nonnormalized weights). The n-th node will receive weighted particles from a certain set of PEs and transmit particles to another (possibly different) set of PEs. To be specific, let us denote

- $\mathcal{N}_n^{\text{in}} \subseteq \{1,2,\dots,N\}$, set of indices corresponding to the nodes that are expected to transmit a subset of their particles *to* the n-th PE, and
- $\mathcal{N}_n^{\text{out}} \subseteq \{1,2,\dots,N\}$, set of indices corresponding to the nodes that expect to receive a subset of the particles generated *at* the n-th PE.

If all the links between PEs are bidirectional, then $\mathcal{N}_n^{\text{in}} = \mathcal{N}_n^{\text{out}}$. For regularity, assume that each PE transmits disjoint subsets of Q particles to each of its designated neighbors. In particular, let

$$\mathcal{M}_t^{n,s} = \{\mathbf{x}_{t-1}^{(n,i_r^s)}, w_{t-1}^{(i_r^s)*}\}_{r=1,\dots,Q}$$

be the particles and weights transmitted from node n to node $s \in \mathcal{N}_n^{\text{out}}$. The indices $i_1^s,\dots,i_Q^s \in \{1,\dots,K\}$ can be selected in any desired way (even randomly) as long as the messages $\mathcal{M}_t^{n,s}$ are disjoint, i.e., $\mathcal{M}_t^{n,s} \cap \mathcal{M}_t^{n,r} = \emptyset$ for any pair $s, r \in \mathcal{N}_n^{\text{out}}$, $s \neq r$. If the messages were not disjoint, then some particles could be replicated and their weights should be modified, so as to keep them proper and guarantee the consistency of the filter approximation. The overall number of particles would also become time-varying. As a consequence, the computational overhead of the algorithm would be increased without any clear advantage.

The information held by the n-th PE *after* the particle exchange at time t is given by $\{\tilde{\mathbf{x}}_{t-1}^{(n,k)}, \tilde{w}_{t-1}^{(n,k)^*}\}_{k=1}^K$ where

$$\{\tilde{\mathbf{x}}_{t-1}^{(n,k)}, \tilde{w}_{t-1}^{(n,k)*}\}_{k=1}^K = \left(\underbrace{\{\mathbf{x}_{t-1}^{(n,k)}, w_{t-1}^{(n,k)*}\}_{k=1}^K}_{\text{initial}} \setminus \underbrace{\left(\bigcup_{s\in\mathcal{N}_n^{\text{out}}} \mathcal{M}_t^{n,s}\right)}_{\text{transmitted}}\right) \cup \left(\underbrace{\bigcup_{s\in\mathcal{N}_n^{\text{in}}} \mathcal{M}_t^{s,n}}_{\text{received}}\right), \tag{5}$$

and we assume that

$$\left|\bigcup_{s\in\mathcal{N}_n^{\text{out}}} \mathcal{M}_t^{n,s}\right| = \left|\bigcup_{s\in\mathcal{N}_n^{\text{in}}} \mathcal{M}_t^{s,n}\right|,$$

for every PE $n \in \{1,\dots,N\}$, so that the number of particles per PE remains constant, $K=M/N$. The new aggregated weight for the n-th node becomes $\tilde{W}_{t-1}^{(n)^*} = \sum_{k=1}^K \tilde{w}_{t-1}^{(n,k)^*}$. Let us remark that the overall sets of particles and weights before and after the particle exchange are identical, i.e.,

$$\bigcup_{n=1}^N \{\mathbf{x}_{t-1}^{(n,k)}, w_{t-1}^{(n,k)*}\}_{k=1}^K = \bigcup_{n=1}^N \{\tilde{\mathbf{x}}_{t-1}^{(n,k)}, \tilde{w}_{t-1}^{(n,k)*}\}_{k=1}^K,$$

while, in general, the aggregated weights are different, $W_{t-1}^{(n)*} \neq \tilde{W}_{t-1}^{(n)*}$.

3.3. Local processing

Immediately after the particle exchange at time t, the weighted particle set at the n-th PE is $\{\tilde{\mathbf{x}}_{t-1}^{(n,k)}, \tilde{w}_{t-1}^{(n,k)*}\}_{k=1,\dots,K}$. The generation of new particles, the update of the importance weights and the resampling step are taken strictly locally, without interaction among different nodes. To be specific, assume that the transition pdf of model (1) is used as an importance function and that the observation vector $\mathbf{y}_t$ is available at every node.[2] Then, at the n-th PE, and for $k=1,\dots,K$,

1. $\overline{\mathbf{x}}_t^{(n,k)}$ is drawn from the pdf $p(\mathbf{x}_t^{(n,k)}|\tilde{\mathbf{x}}_{t-1}^{(n,k)})$, and
2. the corresponding nonnormalized weight is computed as

$$\overline{w}_t^{(n,k)*} = \tilde{w}_{t-1}^{(n,k)*} p(\mathbf{y}_t|\overline{\mathbf{x}}_t^{(n,k)}).$$

Hence the information stored by the n-th node at this point becomes $\{\overline{\mathbf{x}}_t^{(n,k)}, \overline{w}_t^{(n,k)*}\}_{k=1,\dots,K}$ and the aggregated weight is $W_t^{(n)*} = \sum_{k=1}^K \overline{w}_t^{(n,k)*}$.

Next, a resampling step is taken locally by each PE. Assuming a multinomial resampling algorithm,[3] we assign, for $k=1,\dots,K$,

$$\mathbf{x}_t^{(n,k)} = \overline{\mathbf{x}}_t^{(n,j)}, \quad \text{with probability } \overline{w}_t^{(n,j)} \text{ and } j \in \{1,\dots,K\},$$

$$\text{where} \quad \overline{w}_t^{(n,j)} = \frac{\overline{w}_t^{(n,j)*}}{\sum_{l=1}^K \overline{w}_t^{(n,l)*}}, \quad j=1,\dots,K,$$

are the locally normalized importance weights. After resampling, the particles at the n-th PE are equally weighted, namely $w_t^{(n,k)^*} = W_t^{(n)*}/K$. Trivially note that $W_t^{(n)*} = \sum_{k=1}^K \overline{w}_t^{(n,k)*} = \sum_{k=1}^K w_t^{(n,k)*}$, i.e., the resampling step keeps the aggregated weights invariant.

3.4. Estimation

Assume that we are interested in the estimation of moments of the posterior distribution, e.g.,

$$(f,\mu_t) = \int f(\mathbf{x}_t)\mu_t(d\mathbf{x}_t),$$

where f is some function of the state vector at time t, $\mu_t(d\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{y}_{1:t})\, d\mathbf{x}_t$ is the filter probability measure and we introduce the shorthand (f,μ_t) to denote the integral of the function f with respect to the measure μ_t.

We can obtain local estimates of (f,μ_t) at any node. To be specific, we can build discrete random approximations of the measure μ_t at the n-th PE as

$$\overline{\mu}_t^{n,K}(d\mathbf{x}_t) = \sum_{k=1}^K \overline{w}_t^{(n,k)} \delta_{\overline{\mathbf{x}}_t^{(n,k)}}(d\mathbf{x}_t),$$

before the resampling step, and

$$\mu_t^{n,K}(d\mathbf{x}_t) = \frac{1}{K}\sum_{k=1}^K \delta_{\mathbf{x}_t^{(n,k)}}(d\mathbf{x}_t),$$

[2] The availability of the observations, which are typically collected locally in a WSN, involves communications among the nodes. This issue will be addressed in Section 4.

[3] This can be substituted by any other procedure without affecting the rest of the algorithm.

Table 2
Distributed particle filter (DPF) algorithm.

Initialization. At time $t=0$, for $n=1,\ldots,N$:

1. Draw $\mathbf{x}_0^{(n,k)}$, for $k=1,\ldots,K$, from prior $p(\mathbf{x}_0)$.
   Assign $w_0^{(n,k)^*}=\frac{1}{K}$ for all k, set $W_0^{(n)*}=1$.
2. Build the set $\{\mathbf{x}_0^{(n,k)}, w_0^{(n,k)*}, W_0^{(n)*}\}_{k=1}^{K}$.

Recursive step. At time $t>0$, start from the set $\{\mathbf{x}_{t-1}^{(n,k)}, w_{t-1}^{(n,k)*}, W_{t-1}^{(n)*}\}_{k=1}^{K}$. Then, for $n=1,\ldots,N$:

1. **Exchange** particles with neighbor PEs in $\mathcal{N}_n^{\text{in}}$ and $\mathcal{N}_n^{\text{out}}$, as described in Section 3.2, to obtain the sets $\{\tilde{\mathbf{x}}_{t-1}^{(n,k)}, \tilde{w}_{t-1}^{(n,k)^*}\}_{k=1}^{K}$.
2. **Sampling** : Draw $\overline{\mathbf{x}}_t^{(n,k)}$ from $p(\mathbf{x}_t|\tilde{\mathbf{x}}_{t-1}^{(n,k)})$, for $k=1,\ldots,K$.
3. **Weight update:** $\overline{w}_t^{(n,k)*}=\tilde{w}_{t-1}^{(n,k)*}p(\mathbf{y}_t|\overline{\mathbf{x}}_t^{(n,k)})$.
4. **Estimation**: [optional] compute $(f,\overline{\mu}_t^{n,K})$ as in Section 3.4.
5. **Resampling** (local): to obtain the set $\{\mathbf{x}_t^{(n,k)}, w_t^{(n,k)*}, W_t^{(n)*}\}$, where $w_t^{(n,k)*}=W_t^{(n)*}/K$ for every $k=1,\ldots,K$.

after the resampling step, where $\overline{w}_t^{(n,k)}=\overline{w}_t^{(n,k)*}/W_t^{(n)*}$, $k=1,\ldots,K$, are the locally normalized importance weights. This choice of discrete measures leads to the approximations

$$\left(f,\overline{\mu}_t^{n,K}\right)=\sum_{k=1}^{K}\overline{w}_t^{(n,k)}f\left(\overline{\mathbf{x}}_t^{(n,k)}\right)\quad\text{and}$$

$$\left(f,\mu_t^{(n,K)}\right)=\frac{1}{K}\sum_{k=1}^{K}f\left(\mathbf{x}_t^{(n,k)}\right)$$

of the posterior expectation (f,μ_t).

Global estimates can be easily computed by a linear combination of the local estimates. If $W_t^{(n)}=W_t^{(n)*}/\sum_{i=1}^{N}W_t^{(i)*}$ is the globally normalized aggregated weight of the n-th node, then we can build the discrete random measures

$$\overline{\mu}_t^{N,K}(d\mathbf{x}_t)=\sum_{n=1}^{N}W_t^{(n)}\overline{\mu}_t^{n,K}(d\mathbf{x}_t)\quad\text{and}$$

$$\mu_t^{N,K}(d\mathbf{x}_t)=\sum_{n=1}^{N}W_t^{(n)}\mu_t^{n,K}(d\mathbf{x}_t)$$

using the local approximations either before or after resampling, respectively. The resulting global estimates are

$$(f,\overline{\mu}_t^{N,K})=\sum_{n=1}^{N}W_t^{(n)}(f,\overline{\mu}_t^{n,K})\quad\text{and}$$

$$(f,\mu_t^{N,K})=\sum_{n=1}^{N}W_t^{(n)}(f,\mu_t^{n,K}). \tag{6}$$

The resampling operation carried out locally at the N nodes is globally unbiased in the sense defined in, e.g., [15,28]. To make this explicit, consider the sigma algebra generated by the random weights before resampling, i.e., $\mathcal{G}_t=\sigma-(\overline{w}_t^{(n,k)*},\overline{\mathbf{x}}_t^{(n,k)};n=1,\ldots,N;k=1,\ldots,K)$. Since the aggregated weights $W_t^{(n)*}$ are $\mathcal{G}_t-$measurable, for any integrable function f the conditional expectation of the estimator $(f,\mu_t^{N,K})$ given $\mathcal{G}_t$ is

$$\mathbb{E}\{(f,\mu_t^{N,K})|\mathcal{G}_t\}=\sum_{n=1}^{N}W_t^{(n)}\mathbb{E}\{(f,\mu_t^{n,K})|\mathcal{G}_t\} \tag{7}$$

and, since the local normalized weights $\overline{w}_t^{(n,k)}$ and particles $\overline{\mathbf{x}}_t^{(n,k)}$ are also $\mathcal{G}_t-$measurable,

$$\mathbb{E}\{(f,\mu_t^{n,K})|\mathcal{G}_t\}=\sum_{k=1}^{K}\overline{w}_t^{(n,k)}f(\overline{\mathbf{x}}_t^{(n,k)})=(f,\overline{\mu}_t^{n,K}). \tag{8}$$

Substituting (8) into (7) yields

$$\mathbb{E}\{(f,\mu_t^{N,K})|\mathcal{G}_t\}=(f,\overline{\mu}_t^{N,K}),$$

i.e., the DRNA procedure is unbiased.

3.5. Summary

Table 2 summarizes the DPF algorithm investigated in this paper. Note that in order to apply this technique, we assume that *all* the observations in the vector $\mathbf{y}_t$ are available at *every* node at time t. This assumption is fairly natural in the parallel computation setup of [6,28], but not necessarily in the WSN framework of interest here. This issue will be specifically addressed in the subsequent sections. In order to obtain a global estimate of (f,μ_t), each node n in the network should transmit its local estimate $(f,\mu_t^{n,K})$ and its aggregated weight $W_t^{(n)*}$ to a prescribed node (working as a fusion center) where $(f,\overline{\mu}_t^{N,K})=\sum_{n=1}^{N}W_t^{(n)}(f,\overline{\mu}_t^{n,K})$ can be computed. Note that the PEs can keep running asynchronously with the fusion center, i.e., the DPF does not have to be stopped or even delayed to compute the global estimates in (6).

4. Hardware and software framework

4.1. Hardware

In our deployment we use the Imote2; which has CPU set at 23 MHz (although it lacks native support for floating point operations which are, as a result, relatively much slower), 32 Megabytes of memory, and a transceiver with a maximum transmission rate of 31,250 bytes/s. The motes are about $6\times4\times2$ centimeters (including their power source of three AAA batteries).

The hardware limitations define the problem and our approach to it. Our goal is primarily constrained to obtaining the highest possible speed at which each set of observations can be processed (one *sequential step* of the PF) without compromising the mathematical soundness of the algorithm. An increase in the frequency of sequential steps correlates strongly with higher tracking accuracy, and also the speed of the target we are able to track. This takes priority over developing more sophisticated models.

In our deployment, processing is distributed across several nodes, although this becomes a trade-off with the network's capabilities. The DRNA algorithm implies

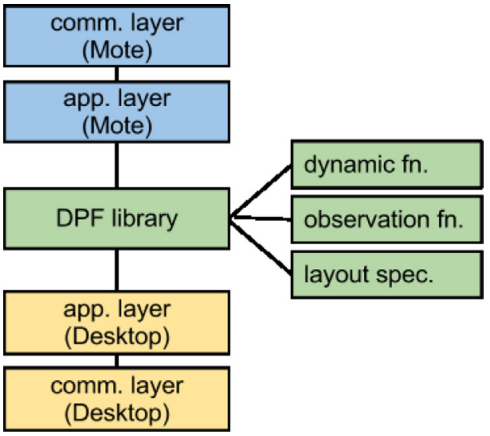

Fig. 1. Our development and testing framework. The DPF library can be used by either a mote (above) or desktop application (below), both interfacing through a communications layer. The latter can be simulated during development so that the entire network can be simulated on the same machine.

a considerable amount of communication between nodes, yet the data transfer between multiple nodes in a wireless sensor network is inherently much less reliable than within multiple processors in a single machine or a wired network. Therefore, balancing the computational and network loads efficiently is the key to providing the best tracking results.

We use the IMOTE2's light sensors, which provide an integer reading between 0 and 65535 relative to the current light level (0 in complete darkness).

4.2. Software

We build the PF described in Section 3 as a library in the C programming language; attractive in terms of speed, and usable on the mote's operating system: TINYOS.[4]

Fig. 1 outlines our framework. To interface our library with the mote hardware, we have developed an *application layer* which calls the appropriate functions and interfaces with the sensor hardware, and a *communications layer* which interfaces the application with the radio. These two layers are written both for the motes for deployment, and in standard C for simulations.

Our distributed PF library implements three scenario-dependent elements: the dynamic function, the observation function, and also the scenario layout specification. The dynamic function describes the motion of the target we are tracking, i.e., the pdf $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. The observation function determines how these observations relate to the state variables, i.e., the pdf $p(\mathbf{y}_t|\mathbf{x}_t)$. The layout specification defines the physical network layout (sensor positions, light source, and the a priori distribution of the state, i.e., the pdf $p(\mathbf{x}_0)$). Any of these elements can be changed to suit different deployments. The source code of our library is available at: https://sourceforge.net/projects/dpflib/files/.

[4] See http://www.tinyos.net

4.3. Implementation

Fig. 2 shows a simple example of distributed network of the type that we consider: a heterogeneous network, consisting of processing elements (PEs) and sensing-only elements (SEs). In our particular deployment, all hardware is identical and both PEs and SEs read and broadcast sensor observations, but only the designated PEs generate particles and compute weights for the DPF. In many settings, the SEs would be much simpler hardware or, all nodes of the network would be PEs.

We assume a fully connected network, within which there are two possible network operations:

1. broadcasting sensor observations to all PEs; and
2. sending particles to specific PEs,

where operation 1 is carried out by all nodes, and operation 2 is carried out only by PEs. With respect to PEs, in practice we find it convenient to combine both operations into a single broadcast packet sent out at each sequential step.

Recall from Section 3.2 that in the particle exchange step of the DPF each PE s, $s \in \{1,\ldots,N\}$, exchanges particles and weights with its neighbors. In particular, the s-th PE receives a set of weighted particles $\mathcal{M}_t^{r,s}$ from every neighbor PE with index $r \in \mathcal{N}_s^{\text{in}} \subset \{1,\ldots,N\}$ and transmits a set of particles $\mathcal{M}_t^{s,k}$ to each PE with index $k \in \mathcal{N}_s^{\text{out}} \subset \{1,\ldots,N\}$. In our implementation we design $|\mathcal{N}_s^{\text{in}}| = |\mathcal{N}_s^{\text{out}}| = 1$, $s = 1,\ldots,N$, for simplicity. In particular, $\mathcal{N}_s^{\text{in}} = \{s-1\}$, for $s = 2,\ldots,N$, and $\mathcal{N}_s^{\text{out}} = \{s+1\}$, for $s = 1,\ldots,N-1$ while $\mathcal{N}_1^{\text{in}} = \{N\}$ and $\mathcal{N}_N^{\text{out}} = \{1\}$.

Since the particle exchange is carried out immediately after the local resampling steps, all samples have equal weights and, therefore, it is enough to select $\mathcal{M}_t^{s-1,s} = \{\mathbf{x}_{t-1}^{(s-1,i)}, w_{t-1}^{(s-1,i)*}\}_{i=1}^{Q}$ and $\mathcal{M}_t^{s,s+1} = \{\mathbf{x}_{t-1}^{(s,i)}, w_{t-1}^{(s,i)*}\}_{i=1}^{Q}$, i.e., each PE transmits its first Q particles and receives a set of Q particles to replace them.

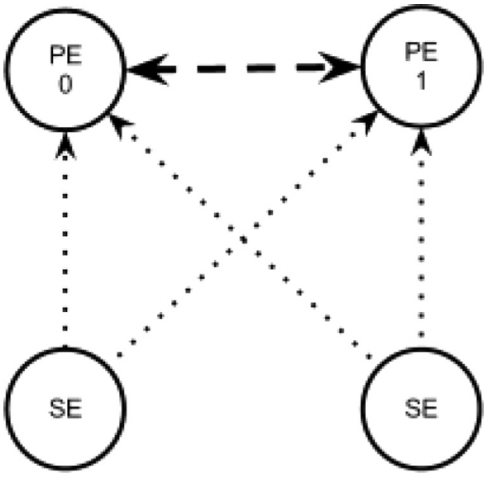

Fig. 2. A small example network layout with four nodes, two of which are designated as PEs and two as SEs. The different thickness of the lines is respective of different amounts of data sent among nodes.

From the point of view of PE s, the following operations are carried out at each time step t:

1. *Comms. layer*: packets arrive from all other nodes $j=\{1,\ldots,J\}\backslash\{s\}$.
2. *Application layer*: extract the observations $y_{t,j}$, $j=\{1,\ldots,J\}\backslash\{s\}$; extract particles received from the PE $n\in\mathcal{N}_s^{\text{in}}$, namely the set $\mathcal{M}_t^{n,s}=\{\mathbf{x}_{t-1}^{(n,i)},w_{t-1}^{(n,i)*}\}_{i=1}^{Q}$.
3. *DPF library*: create the observation vector $\mathbf{y}_t=\{y_{t,1},\ldots,y_{t,J}\}^\top$ and the set $\{\tilde{\mathbf{x}}_{t-1}^{(s,k)},\tilde{w}_{t-1}^{(s,k)*}\}_{k=1}^{K}$ according to Eq. (5).
4. *DPF library*: step through the PF (recursive steps 2–5 of Table 2) to obtain the set $\{\mathbf{x}_t^{(s,k)},w_t^{(s,k)*}\}_{k=1}^{K}$.
5. *Application layer*: wait for timer tick, then read the observation $y_{t+1,s}$ from the sensor and put it into a packet with the message $\mathcal{M}_{t+1}^{s,n}=\{\mathbf{x}_t^{(s,i)},w_t^{(s,i)*}\}_{i=1}^{Q}$, where $n\in\mathcal{N}_s^{\text{out}}$.
6. *Comms. layer*: broadcast the packet.

The computation of estimates is not, strictly speaking, part of the algorithm. Local estimates can be computed at any time in the PEs without interrupting the flow of the algorithm. Specifically, given some integrable function of interest f, the approximate integral $(f,\mu_t^{s,K})$ can be computed in the s-th PE after step (3) is completed. In step (5) the pair of $\{(f,\mu_t^{s,K}),W_t^{(s)*}\}$ can be wrapped into the packet to be transmitted and then broadcast in step 6. One or more PEs in the network collect the local estimates $(f,\mu_t^{n,K})$, $n=1,\ldots,N$, and compute the overall estimate $(f,\mu_t^{N,K})=\sum_{n=1}^{N}W_t^{(n)}(f,\mu_t^{(n,K)})$. In our setup, the task is carried out by PE 1, but this is completely arbitrary.

The network is initialized by generating a command broadcast from a base node. Upon receiving this command, all motes set their timers to the specified frequency and initialize them. The time count is set to $t=0$ and initial particle sets are drawn from the prior $p(\mathbf{x}_0)$, i.e., the sets $\{\mathbf{x}_0^{(n,k)}\}_{k=1}^{K}$ are generated, with aggregated weights $W_0^{(n)*}=1$ for every n and $w_0^{(n,k)}=1/K$ for every n and k. At time $t=1$, each node transmits its sensor observation $y_{1,j}$, $j=1,\ldots,J$, which begins the loop (no particle exchange is actually needed at time $t=0$).

In our experiments, the fusion of local estimates into a global estimate takes place on the designated mote at each timestep, as soon as the incoming packet is unpacked, i.e., as soon as the other motes' local estimates are available. Depending on the application, this estimate could be recorded (for later analysis) or forwarded on to be monitored by the user.

5. Construction of observation models

The goal is to implement a DPF for target tracking using WSN that collects light intensity measurements. Our experimental scenario is depicted in Fig. 3. It is a room with $J=10$ nodes (each equipped with a light sensor) enclosing an area of 3.2×6.0 m with a single source of natural light (a window). Although there is little variation in artificial light, natural light comes more readily from side-on, making it more practical to carry out indoor tracking without modification to the existing setup of the room (by adding lamps, etc.).

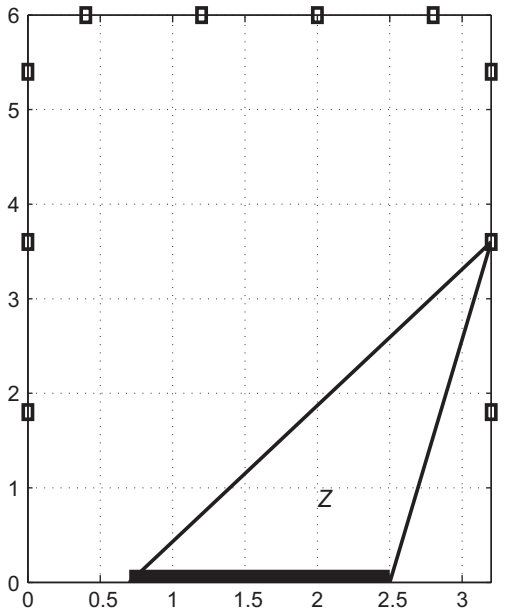

Fig. 3. Tracking scenario of 3.2×6.0 m (a top-down view). The thick line is the light source (a window). There are $J=10$ motes equipped with light sensors around the edges, indicated by squares. The entry to the scenario lies at the bottom-right corner. The detection zone (Z) is exemplified for one of the sensors.

Using light sensors for position estimation is fundamentally different from using other types of observations such as RSS, sound waves and GPS signals, which are essential functions of the distance between the target and the sensor. Light sensors only provide an integer value proportional to the amount of light they are receiving. In a uniform medium (such as, approximately, the air inside a room), light travels in straight lines, and over short distances (e.g., several meters, as we are considering) scattering from suspended particles and air molecules has a negligible effect on light. A good review of optics is given in [24]. Hence, any object geometrically enclosed in the region between a light sensor and a light source can affect the values being read by that sensor. If the source of light is a window, in a top-down two-dimensional representation the detection zone can be viewed as a triangle, as depicted in Fig. 3 (i.e., the Z region).

Reflections and scattering from surfaces (such as walls), inanimate objects (e.g., furniture) and the target itself are far too complex to be modeled, especially on our limited hardware. Since it is very hard to translate the disturbances caused by the target in the sensor readings into distance measurements, we instead focus on obtaining binary observations: 1 if the target is inside the detection zone and 0 otherwise.

As the target keeps moving within the detection zone, it may block light or actually reflect light into the light sensor, thus causing the light level reading to go either up

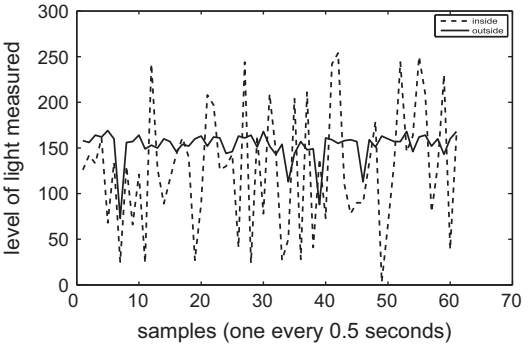

Fig. 4. The light signal when the target (a walking person) moves randomly outside the detection zone (solid line) and randomly inside the detection zone (dashed line). Rather than a reduction in the light level, the presence of a target within the detection zone causes a large variance in the sensor readings.

or down. This effect is illustrated in Fig. 4. Furthermore, under natural light there is no base level for the measured signal (when no target is present) due to changes in the weather or the time of day. Therefore, it is very difficult to detect the target presence from the mean or instantaneous amplitude of the light signal. Instead, the short-term variance of light readings is very informative and becomes a reliable indicator of the presence of moving objects in the detection zone. Note that a motionless target becomes 'invisible' (as the variance of light readings stabilizes) but its position is not 'lost' by the tracker.

In order to turn this intuition into a quantitative model, we propose to process the light readings at every node (both PEs and SEs) in order to convert them into binary data. Between the sequential steps of $t-1$ and t each sensor $j=1,\ldots,J$ collects L light readings (evenly spaced in time) to produce a set $V_{t,j}=\{v_{1,t,j},\ldots,v_{L,t,j}\}$ (with $L=5$ in our experiments). For a given threshold τ, the j-th sensor outputs the binary datum

$$y_{t,j}=\begin{cases}1 & \text{if } \mathrm{var}(V_{t,j})>\tau\\ 0 & \text{otherwise}\end{cases} \tag{9}$$

where $\mathrm{var}(V_{t,j})$ is the empirical variance of the sample $V_{t,j}$. Putting the observations of all nodes together into a single $J\times 1$ vector we obtain, for every timestep t, the full observation $\mathbf{y}_t=[y_{1,t},\ldots,y_{J,t}]^\top$.

For our experiments, we calibrated the threshold τ a priori (offline) from a set of real-world data. Specifically, we instructed a person (acting as the target) to walk following an arbitrary trajectory within the monitored area during a period comprising T discrete time steps. At each time step $t=1,\ldots,T$, each sensor j collected samples $V_{t,j}=\{v_{1,t,j},\ldots,v_{L,t,j}\}, j=1,\ldots,J$ (one set per sensor). With the data collected up to time T, we selected the threshold τ as the average of the empirical variances of all sensors, i.e.,

$$\tau=\frac{1}{JT}\sum_{j=1}^{J}\sum_{t=1}^{T}\mathrm{var}(V_{t,j}).$$

The same threshold is then used by both PEs and SEs.

Recall from Section 3 that the nonnormalized weight for the k-th particle of the n-th PE is computed as

$$\overline{w}_t^{(n,k)*}=\tilde{w}_{t-1}^{(n,k)*}p(\mathbf{y}_t|\overline{\mathbf{x}}_t^{(n,k)}).$$

Therefore, it is necessary to specify a model for the likelihood $p(\mathbf{y}_t|\mathbf{x}_t)$. We assume that the observations are conditionally independent[5] across the different sensors, hence $p(\mathbf{y}_t|\mathbf{x}_t)=\prod_{j=1}^{J}p(y_{j,t}|\mathbf{x}_t)$ and we only need to specify the marginal density $p(y_{j,t}|\mathbf{x}_t)$. We define the latter as

$$p(y_{j,t}|\mathbf{x}_t)=\begin{cases}1-F^+ & \text{if } \mathbf{x}_t\in Z_j \text{ and } y_{j,t}=1\\ F^+ & \text{if } \mathbf{x}_t\notin Z_j \text{ and } y_{j,t}=1\\ 1-F^- & \text{if } \mathbf{x}_t\notin Z_j \text{ and } y_{j,t}=0\\ F^- & \text{if } \mathbf{x}_t\in Z_j \text{ and } y_{j,t}=0\end{cases} \tag{10}$$

where Z_j is the two-dimensional detection zone enclosed by sensor j and the vertices of the light source, and F^+ and F^- are the *false positive* and *false negative* rates associated with the deployment (i.e., the probability of $y_{j,t}=1$ when the target is outside Z_j, and the probability of $y_{j,t}=0$ when the target is inside Z_j, respectively). The error rates F^+ and F^- are calibrated empirically from a short supervised walk ($T=20$ steps) in the real-world scenario. Specifically, we set

$$F^+=\frac{1}{JT}\sum_{j=1}^{J}\sum_{t=1}^{T}I\left(y_{j,t}=0,\mathbf{x}_t\in Z_j\right),$$

$$F^-=\frac{1}{JT}\sum_{j=1}^{J}\sum_{t=1}^{T}I\left(y_{j,t}=1,\mathbf{x}_t\notin Z_j\right),$$

where each $y_{t,j}$ is given from Eq. (9) on the motes and $I(a,b)$ is an indicator function returning 1 if, and only if, both a and b are true, and 0 otherwise. In our experimental setup, this supervised walk was an X-shaped trajectory passing through all sensors' detection zones at least once. The error probabilities F^- and F^+ are constants in the model and identical for all PEs. The (supervised and a-priori known) walk used to calibrate F^+,F^- is different from the (unsupervised and arbitrary) walk used to obtained the threshold τ.

Note that the constants in the likelihood model (i.e., τ, F^+, and F^-) are the same for all nodes. It is possible to select a unique threshold τ_j (and, hence, unique error rates F_j^+ and F_j^-) for every sensor by using supervised trajectories in and out of the detection zones $Z_1,\ldots,Z_J$. This is intuitively quite appealing from the point of view of trying to maximize the accuracy of the observation model. However, we found that in real-world tests this approach leads to worse performance than a single general model for all nodes. This is probably due to the greater possibilities of introducing errors when manually aligning sensor readings with the true trajectory (necessary for a supervised walk), and from inadvertently obtaining overly 'sterile' data by choreographing the target's movements too precisely, as well as overfitting the data (the target used for training will not be the same one for testing). Moreover, manually aligning the binary sensor readings

[5] This means that the observations are independent conditional on a given target position. This is a rather common assumption. Intuitively, it means that the observational noise at different sensors is independent.

for the true trajectory is a very intensive task, impractical for many real-world deployments. In light of this, we decided to calibrate a single observation model, common to all sensors.

6. Simulation and experimental results

We illustrate the validity of our approach by applying the proposed DPF algorithm in a real-world WSN for target tracking using the binary observation model described in Section 5. We first describe the dynamic model for the target, then show results both for synthetic and experimental (real-world) observations. We conclude the section with a brief discussion of the advantages and disadvantages of the proposed scheme.

6.1. Setup

We consider a state-space vector $\mathbf{x}_t = [x_{1,t}, x_{2,t}, x_{3,t}, x_{4,t},]^\top \in \mathbb{R}^4$ that describes the position $\mathbf{p}_t = [x_{1,t}, x_{2,t}]^\top$ and velocity $\mathbf{v}_t = [x_{3,t}, x_{4,t}]^\top$ of the target at time t.

The prior distribution $p(\mathbf{x}_0)$ is (multivariate) Gaussian. In particular, $\mathbf{p}_0$ and $\mathbf{v}_0$ are a priori independent and

$$p(\mathbf{p}_0) = N(\mathbf{p}_0|\overline{\mathbf{p}}_0, \sigma_{p,0}^2\mathbf{I}_2) \quad \text{and} \quad p(\mathbf{v}_0) = N(\mathbf{v}_0|\overline{\mathbf{v}}_0, \sigma_{v,0}^2\mathbf{I}_2),$$

where $N(\mathbf{z}|\overline{\mathbf{z}}, \mathbf{\Sigma})$ denotes the Gaussian density of the random vector $\mathbf{z}$, with mean $\overline{\mathbf{z}}$ and covariance matrix $\mathbf{\Sigma}$, $\mathbf{I}_2$ is the 2×2 identity matrix and

- the mean prior position is $\overline{\mathbf{p}}_0 = [2.5, 0.4]^\top$, i.e., the target enters through the bottom-right corner of the area of interest sketched in Fig. 3(left),
- the mean prior velocity is $\overline{\mathbf{v}}_0 = [-0.2, 0.2]^\top$, i.e., the target is initially expected to move toward the middle of the area of interest, and
- the prior variances are $\sigma_p^2 = 0.5$ and $\sigma_v^2 = 4\times 10^{-3}$.

Assume that the region of interest is a rectangle of the form $A = A_1 \times A_2$, where $A_1 \in \mathbb{R}^2$ and $A_2 \in \mathbb{R}^2$. Then we model the target dynamics as

$$\mathbf{x}_t = \mathbf{b}_A(\mathbf{f}(\mathbf{x}_{t-1}, a_t) + \mathbf{u}_t), \tag{11}$$

where

- $\{a_t\}_{t\geq 1}$ is an i.i.d. sequence of indicator random variables with pmf $p(a_t = 1) = \alpha_1 = 0.1$ (and $p(a_t = 0) = 1-\alpha_1 = 0.9$),
- $\mathbf{f}(\mathbf{x}_{t-1}, a_t)$ is a vector-valued state transition function (specified below),
- $\mathbf{u}_t$ is the process noise, with density $N(\mathbf{u}_t|\mathbf{0}, \mathbf{C}_u)$ and $\mathbf{C}_u = \begin{bmatrix} \sigma_p^2\mathbf{I}_2 & \mathbf{0} \\ \mathbf{0} & \sigma_v^2\mathbf{I}_2 \end{bmatrix}$,
- and $\mathbf{b}_A$ is a 'wrapper' function designed to keep the target motion within the limits of the region A (also described below).

The indicator a_t determines the kind of motion of the target. If $a_t = 0$, then $\mathbf{f}_t(\cdot, 0)$ yields a constant-velocity [4] model. If, on the other hand, $a_t = 1$, then $\mathbf{f}(\cdot, 1)$ produces a sharp turn by generating a velocity vector independent of the velocity at time $t-1$, specifically:

$$\mathbf{f}(\mathbf{x}_{t-1}, 0) = \begin{bmatrix} 1 & 0 & T_s & 0 \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \end{bmatrix} \quad \text{and}$$

$$\mathbf{f}(\mathbf{x}_{t-1}, 1) = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ v_t\ \cos(\omega_t) \\ v_t\ \sin(\omega_t) \end{bmatrix}$$

where T_s is the time discretization period, v_t is the modulus of the velocity at time t, drawn from the uniform pdf $p(v_t) = \mathcal{U}(0, V_{max})$, and ω_t is the angle of the velocity at time t, drawn from the uniform pdf $p(\omega_t) = \mathcal{U}(0, 2\pi)$. The maximum velocity is set to $V_{max} = 1.5$ m/s.

In simulations the target will reverse any of its velocity components when moving out of the scenario bounds (intuitively, it will 'bounce' off the walls). This reflects the fact that target motion is restricted by the bounds of an indoor scenario. In particular, recalling that $A = A_1 \times A_2$, we define the wrapper function $\mathbf{b}_A$ as

$$\mathbf{b}_A\begin{pmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \end{pmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \times (-1)^{I(x_{1,t}\notin A_1)} \\ x_{4,t} \times (-1)^{I(x_{2,t}\notin A_2)} \end{bmatrix}$$

where the indicator function $I(x_d \notin A_d)$ returns 1 if and only if $\mathbf{x}_d$ falls outside the interval A_d, $d \in \{1, 2\}$.

Table 3 displays the values of the relevant simulation and algorithm parameters. Note that the number of PEs N is the only variable which we change directly in our experiments. For example, we use $N = 1$ as the equivalent to a centralized PF (for comparison). Note, however, that changing N affects other variables, such as the number of SEs ($J-N$) and the number of particles per PE ($K = M/N$). It does not matter which of the nodes are PEs and which are SEs, since we assume a fully connected network. Each node (either PE or SE) produces one binary observation ($y_{j,t}$) every T_s seconds.

Each PE transmits $Q = 1$ particles in the particle exchange step. Recall that the exchange is carried out in a circular manner, (see Section 4.3). Local resampling is

Table 3
DPF and model parameters.

Variable	Symbol	Value (unit)
No. of PEs	N	4[a]
No. of nodes	J	10
No. of SEs		$J-N$
Total no. of particles	M	100
No. of particles/PE	K	M/N
No. of exchanged particles	Q	1
No. of timesteps	T	18
Sampling period	T_s	1.0 (s)
Position variance	σ_p^2	0.5 (m^2)
Velocity variance	σ_v	0.004 (m^2)
Maximum velocity	V_{max}	1.5 (m/s)

[a] Varied for some experiments.

carried out at each step, which keeps the computational load even for all t and eliminates the overhead of checking if resampling is necessary. We use systematic resampling [8] which is significantly faster than, e.g., multinomial resampling schemes.

Table 4
The processing time for DPF for various values of N in number of seconds per timestep and the equivalent number of timesteps (i.e., observations) per minute that each filter can handle. The total number of particles is constant ($M=100$); 100/N per PE. Note that this time does *not* include network activity.

Processing	CPF ($N=1$)	DPF ($N=2$)	DPF ($N=4$)	DPF ($N=8$)
Sec./timestep	3.37	1.51	0.73	0.26
Timesteps/min.	17.80	39.74	82.19	230.77

6.2. Computer simulations

We compare the proposed DPF with a standard CPF over 100 random and independent target trajectories $\mathbf{x}_{0:T}$, with associated synthetic data $\mathbf{y}_{1:T}$ drawn from the pmf $p(\mathbf{y}_t|\mathbf{x}_t)$ specified in Section 5. Then we applied both the CPF ($N=1$) and the DPF (with $N=4$ PEs) with the same total number of particles M to track each sample trajectory from the associated sequence of synthetic observations. We simulate with one observation ($\mathbf{y}_t$) per second ($T_s = 1$), which is what the DPF $N=4$ can obtain in practice, but

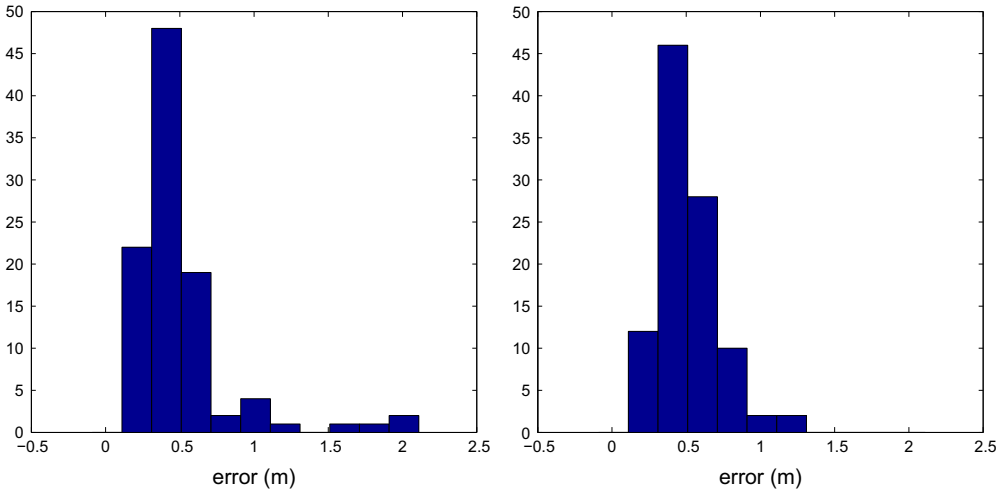

Fig. 5. Histogram of the position error in meters for both the centralized (left) and distributed (right) versions of the PF over 100 simulated trajectories. In both cases the error is about half a meter on average.

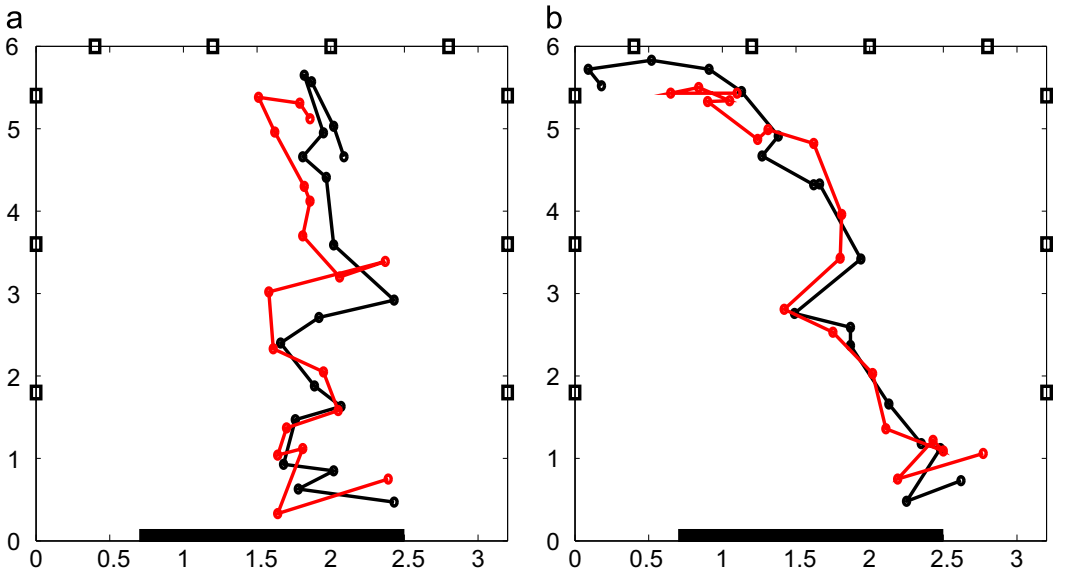

Fig. 6. The simulated (black) paths for two (of the 100) simulations, and the corresponding DPF-estimated paths (red); each over $T=18$ timesteps. (a) Run 1. (b) Run 2. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

note that it is impossible for the CPF to obtain this performance in practice on this hardware (the observation rate is too high for a single mote to cope with; see Section 6.3 and Table 4 for a discussion). The purpose of this simulation is to compare the performance of a DPF with that of an ideal CPF free of practical constraints, i.e., how much performance is lost on account of distributing the computation among the nodes.

Fig. 5 displays the empirical distribution of errors, and the average error, for 100 simulated paths. For a sequence of position estimates $\hat{\mathbf{p}}_t = [\hat{x}_{1,t}, \hat{x}_{2,t}]^\top$, $t = 1, \ldots, T$, the absolute error at each step t is

$$\varepsilon_t = \sqrt{(x_{1,t} - \hat{x}_{1,t})^2 + (x_{2,t} - \hat{x}_{2,t})^2},$$

and the mean absolute error (MAE) for a trajectory is $\overline{\varepsilon} = 1/T \sum_{t=1}^{T} \varepsilon_t$. The CPF obtains an average MAE for 100 paths of 0.4991 m, and the DPF obtains 0.5115. We see that the performance loss in the DPF is minimal.

Fig. 6 plots a selection of these paths along with the path estimated by our DPF. The discrepancies between true and estimated location tend to occur when the target moves between detection zones. As the observations are binary and zone-based, rather than distance-based, there are 'gaps' around the edges (see for example the final points in Fig. 6b). Accuracy also tends to be higher nearer the light source where more detection zones overlap (see for example, Run 1).

Fig. 8 shows the evolution of the total weight within each node during simulation – both exchanging particles at each step and every five steps. The benefit of exchanging particles is clearly observable.

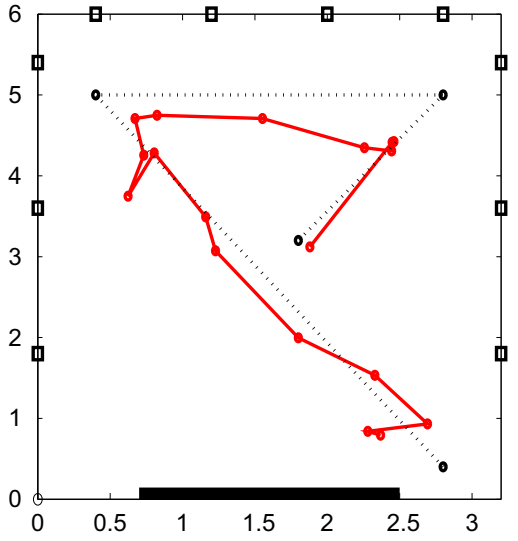

Fig. 7. Results of the DPF (solid line, $N=4$) tracking a target walking a prescribed trajectory (dashed line). Of the three straight lines making up the true path, each is walked slightly faster than the previous one, with a pause of about 1 s taken at the point where the direction is changed (indicated by hollow circles). The path is walked in $T=18$ timesteps (18 s in our setup).

6.3. Experimental results

Using the parameter values in Table 3 we carried out the following experiment to track a person walking across the monitored region.

1. The person was instructed to walk a prescribed path through our real-world scenario (dashed line in Fig. 7) with each segment being slightly faster than the last. The trajectory lasts $T=18$ s. Each node generated one observation per second, $y_{j,t}$, $j = 1, \ldots, 10$, $t = 1, \ldots, 18$.
2. As the person walked, we run the DPF with $N=4$ PEs in *real-time* to process the observations $\mathbf{y}_t$, $t = 1, \ldots, T$, and produce estimates $\hat{\mathbf{x}}_t = \sum_{n=1}^{N} W_t^{(n)} \sum_{k=1}^{K} \mathbf{x}_t^{(n,k)} \overline{w}_t^{(n,k)}$, $t = 1, \ldots, T$, of the target trajectory.

Fig. 7 shows the trajectory the person was asked to walk, and the path estimated by the DPF. The true path can only be drawn approximately, although we expect it to be accurate to within a fraction of a meter. Time points are unavailable, as we have no way of accurately synchronizing the target movements with the scenario, but we know the target walked each of the three segments slightly faster than the previous one and the full path took 18 s to complete. As we expected, the target made brief pauses of about 1 s at the point where the direction changed (this

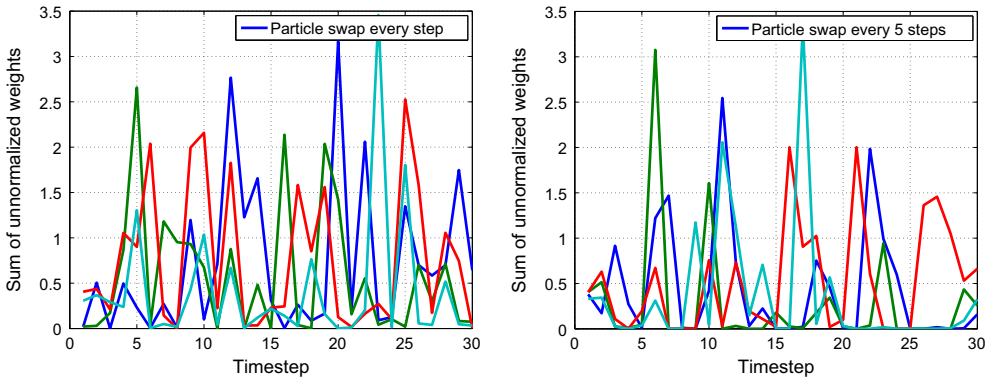

Fig. 8. These plots show the evolution of the sum of nonnormalized particle weights for 4 motes over 30 timesteps, both for particle-exchange every timestep (left) and every five timesteps (right). Nodes do better with a regular particle exchange, seen on the right *after* steps 5, 10, 15, etc. The view is a bit noisy due to the relatively small scenario space and the 'jumpy' likelihood function, but it is clear that the particle exchange has a positive effect.

Table 5
The accuracy of a CPF under a decreasing number of observations per second (corresponding to an increasing sampling period T_s), over 20 timesteps (observations $\mathbf{y}_{1:20}$). The corresponding MAE has been averaged over 100 simulations. Note that $\mathbf{y}_{1:20}$ covers a larger trajectory for larger T_s (because an observation is processed every T_s seconds). The table shows the potential performance improvement by distributing processing with a DPF. Recall that, in a real-world deployment with DPF $N=4$, $T_s=1$ (highlighted in the table) can be held.

T_s	0.5	**1.0**	1.5	2.0	2.5	3.0	3.5	4.0
MAE (m)	0.31	0.50	0.77	1.33	1.48	1.63	1.77	1.93

reflects the time necessary to stop and change direction, rather than a scripted break).

The DPF performs comparably to the CPF in the simulations and, although it may have some difficulty near the edges, where there are fewer detection zones, it quickly catches up when the target crosses the detection zones again. The experimental trajectory is challenging because three of the four points are right on the edge (if not slightly outside) the detection zones. At the final point, where several detection zones overlap, the target is estimated within centimeters.

6.4. Performance study

The speed (and also the accuracy) of the DPF is influenced by a trade off between processing and network communications. We study both of these factors separately.

Table 4 displays the average processing time required *per timestep* of DPFs with $N=1, 2, 4, 8$ ($N=1$ corresponding to the CPF) and a constant number of particles $M=100$ shared among all PEs ($K=M/N$ each[6]) on the earlier-described real-world trajectory. We see that halving N approximately doubles the running time per timestep. Clearly the 3.37 s per timestep as obtained by the CPF ($N=1$) is not adequate for real-time tracking. Over a run of 18 s, this only allows for five steps through the PF. Note that a CPF simulated on an Intel Xeon 3.16 GHz CPU runs at 0.0008 per timestep (over 4000 times faster); a clear indication of the hardware limitations we are dealing with, as well as the efficiency of our implementation.

Table 5 displays the simulated performance for different sampling periods of the CPF. We already saw in Table 4 that any period less than about $T_s=3.5$ is unobtainable in practice (namely, $T_s=3.37$ without including network overhead). The error at $T_s=3.5$ is three times greater than at $T_s=1$, which can be managed with the proposed DPF in practice (see Section 6.3). Since we have already showed that the performance of the CPF and DPF are comparable given the same T_s (see Section 6.2), we can say that in our real-word testbed, the DPF obtains three times greater accuracy compared to a practical CPF. Furthermore we note that, although the loss appears to taper a bit at 3.0 and 4.0 s per timestep, this is almost certainly due to the bounds of the scenario (3.3×6.0; thus 2.0 m is about as much error as one can incur).

[6] Or as close to it as possible: for $N=8$ we round down to $K=12$.

Table 6
The *network activity* (packets and bytes) per timestep for J motes of N PEs and $J-N$ SEs. We store each 4-dimensional state $\mathbf{x}_t$ with its weight w_t in 20 bytes (4 bytes for each number) and each observation y_t in 1 byte.

Network load	CPF ($N=1$)	DPF ($N\geq 2$)	DPF ($N\geq 2$ with global est.)
No. of packets	$J-1$	J	J
No. of bytes	$J-1$	$J+20N$	$J+40N+8$[a]

[a] Only estimating the *position*-coordinates of the state (2×4 bytes).

Table 7
Memory usage for M particles, a state size of $d=4$, and N PEs. Assuming 4 bytes to store floating point values (as is the case on the iMote2).

Variable	Memory (bytes)
Weights	$4M$
Normalized weights	$4M$
States	$4dM$
States-buffer (used in resampling)	$4dM$
Estimations (local, global, norm. const.)	$4d+4dN+4N$
Layout, observation, constants, misc.	100 (approx.)

Table 6 shows how the network communication load increases with more PEs. All DPFs in this layout send only one more packet per timestep than a CPF, since all nodes must broadcast their observations at each timestep (to the CPF's single PE). On the other hand, the number of bytes per timestep increases linearly with respect to the number of PEs (N), as particles must be shared among them. With $N=4$ this equates to 90 bytes (or 178 if needing to broadcast a global estimate) at each timestep, for which we have up to $T_s-0.73=0.27$ s (refer to Tables 3 and 4 for $N=4$). At the iMote2s' maximum bandwidth of 31,250 bytes/s, this is easily doable (even if we take into account a real-world scenario where this maximum is not achievable, packet overhead, time taken to resend dropped packets, etc., we are still left with a 'comfortable' margin).

Table 7 shows the main memory usage by the DPF implementation; with $N=4$ PEs ($K=25$ particles per PE) we use up to 4196 bytes per mote, a tiny fraction of the sciMote2's 32 Megabytes.

In the implementation we have presented, some optimization is still possible for faster performance. For example storing the state as integers with only 2 bytes instead of 4, and using a single bit to store individual observations, and only broadcast it if it is 1. This would reduce network traffic and possibly also computational time. However, we did not wish to present an over-optimized system for a single scenario, but rather a generic one that is suitable for deployments in a variety of environments.

We have not measured battery consumption directly (due to the difficulty in doing so on the iMote2). Although we have not needed to replace the batteries throughout several hours of testing, clearly our testbed network could not run continuously for weeks on end, as it requires a relatively large amount of processing and radio traffic as compared to many other WSN applications. Nevertheless,

it would be trivial to put the network in a sleep mode while no activity was detected by any sensors.

7. Conclusions and future work

We have described the implementation of a distributed particle filter (DPF) for target tracking in a wireless sensor network. Unlike existing work, the proposed method guarantees that the particle weights are constructed properly, and hence also the state estimators. We have carried out a series of simulations using models fitted with real light intensity data that show a tracking precision of around half a meter. In this respect, the accuracy difference between the proposed DPF and a centralized filter with the same total number of particles is less than 2 cm, whereas only the distributed version is fast enough for real-world deployment on the hardware we consider. To support this claim we have implemented a real-world WSN to track a moving target in a 3.2×6.0 m indoor scenario using only light-intensity measurements; accuracy is about half a meter on average.

The DPF with four processing nodes is over four times faster than an equivalent centralized version, meaning, equivalently, that the same performance can be obtained on less powerful hardware. A greater proportion of processing nodes does imply more reliance on efficient communications, but applications are mainly limited only by the overall size of the network (since all observations must be shared between all nodes). Adaptations to our filter will be needed to scale up to much larger networks with higher dimension observations; for example, by only requiring observations from an active area, or working with out-of-sequence observations. There is no reason to believe that such adaptations are not possible, and we have shown that our particle filter is already suitable to wireless sensor network scenarios, on hardware thousands of times slower than a typical desktop machine.

References

[1] K. Achutegui, L. Martino, J. Rodas, C.J. Escudero, J. Miguez, A multi-model particle filtering algorithm for indoor tracking of mobile terminals using RSS data, IEEE Control Appl. (CCA) Intell. Control (ISIC) 2009.

[2] N. Ahmed, Y. Dong, T. Bokareva, S. Kanhere, S.Y. Jha, T. Bessell, M. Rutten, B. Ristic, N. Gordon, Detection and tracking using wireless sensor networks, in: 5th International Conference on Embedded Networked Sensor Systems SenSys '07, ACM, 2007, pp. 425–426.

[3] A. Bain, D. Crisan, Fundamentals of Stochastic Filtering, Springer, 2008.

[4] Y. Bar-Shalom, X.R. Li, (Eds.), Estimation with Applications to Tracking and Navigation, Wiley & Sons, 2001.

[5] Tatiana Bokareva, Wen Hu, Salil Kanhere, Branko Ristic, Travis Bessell, Mark Rutten, Sanjay Jha. Wireless sensor networks for battlefield surveillance, in: Land Warfare Conference, 2006.

[6] M. Bolić, P.M. Djurić, S. Hong, Resampling algorithms and architectures for distributed particle filters, IEEE Trans. Signal Process. 53 (July (7)) (2005) 2442–2450.

[7] Claudio J. Bordin, Marcelo G.S. Bruno, Cooperative bling equalization of frequency-selective channels in sensor networks using decentralized particle filtering, in: 42nd Asilomar Conference on Signals, Systems and Computers, October 2008, pp. 1198–1201.

[8] J. Carpenter, P. Clifford, P. Fearnhead, Improved particle filter for nonlinear problems, IEE Proc.—Radar Sonar Navig. 146 (February (1)) (1999) 2–7.

[9] Daizhan Cheng, Bijoy Ghosh, Xiaoming Hu, Distributed sensor network for target tracking, in: 17th International Symposium on Mathematical Theory of Networks and Systems, 2006.

[10] Mark Coates, Distributed particle filters for sensor networks, in: The International Conference on Information Processing in Sensor Networks, (IPSN), April 2004, pp. 99–107.

[11] A. Dhital, P. Closas, C. Fernández-Prades, Bayesian filtering for indoor localization and tracking in wireless sensor networks, EURASIP J. Wireless Commun. Networking 21 (2012), http://dx.doi.org/10.1186/1687-1499-2012-21.

[12] P.M. Djurić, J.H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M.F. Bugallo, J. Míguez, Particle filtering, IEEE Signal Process. Mag. 20 (September (5)) (2003) 19–38.

[13] P.M. Djurić, Ting Lu, M.F. Bugallo, Multiple particle filtering, in: 32nd IEEE ICASSP, April 2007.

[14] P.M. Djuric, M. Vemula, M.F. Bugallo, Target tracking by particle filtering in binary sensor networks, IEEE Trans. Signal Process. 56 (June) (2008) 2229–2238.

[15] R. Douc, O. Cappé, E. Moulines, Comparison of resampling schemes for particle filtering, in: 4th International Symposium on Image and Signal Processing and Analysis, September 2005, pp. 64–69.

[16] A. Doucet, N. de Freitas, N. Gordon (Eds.), Sequential Monte Carlo Methods in Practice, Springer, 2001.

[17] A. Doucet, S. Godsill, C. Andrieu, On sequential Monte Carlo sampling methods for Bayesian filtering, Stat. Comput. 10 (3) (2000) 197–208.

[18] Donald P. Eickstedt, Cooperative target tracking in a distributed autonomous sensor network, in: Proc. IEEE of OCEANS, 2006, pp. 1–6, http://dx.doi.org/10.1109/OCEANS.2006.306976.

[19] N. Gordon, D. Salmond, A.F.M. Smith, Novel approach to nonlinear and non-Gaussian Bayesian state estimation, IEE Proc. FRadar Signal Process 140 (1993) 107–113.

[20] O. Hlinka, F. Hlawatsch, P. Djuric, Distributed particle filtering in agent networks, IEEE Signal Process. Mag. (January) (2013) 61–81.

[21] O. Hlinka, O. Sluciak, F. Hlawatsch, P. Djuric, M. Rupp, Distributed Gaussian particle filtering using likelihood consensus, in: International Conference on Acoustics, Speech and Signal Processing, May 2011, pp. 3756–3759.

[22] H. Huo, Y. Xu, H. Yan, S. Mubeen, H. Zhang, An elderly health care system using wireless sensor networks at home, in: 2009 Third International Conference on Sensor Technologies and Applications SENSORCOMM '09, IEEE Computer Society, 2009, pp. 158–163.

[23] R.E. Kalman, A new approach to linear filtering and prediction problems, J. Basic Eng. 82 (1960) 35–45.

[24] N. Kumar, Comprehensive Physics XII, Laxmi Publications, 2008.

[25] A. Lee, C. Yau, M.B. Giles, A. Doucet, C.C. Holmes, On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods, J. Comput. Gr. Stat. 19 (4) (2010) 769–789.

[26] H.Q. Liu, H.C. So, F.K.W. Chan, K.W.K. Lui, Distributed particle filter for target tracking in sensor networks. Prog. Electromagnet. Res. C 11 (2009) 171–182.

[27] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, J. Anderson, Wireless sensor networks for habitat monitoring, in: 1st ACM International Workshop on Wireless Sensor Networks and Applications WSNA '02, ACM, 2002, pp. 88–97.

[28] J. Míguez, Analysis of parallelizable resampling algorithms for particle filtering, Signal Process. 87 (12) (2007) 3155–3174.

[29] R. Olfati-Saber, Distributed Kalman filtering for sensor networks, in: 46th IEEE Conference on Decision and Control, December 2007, pp. 5492–5498.

[30] A. Ribeiro, G.B. Giannakis, S.I. Roumeliotis, SOI-KF: distributed Kalman filtering with low-cost communications using the sign of innovations, IEEE Trans. Signal Process. 54 (December (12)) (2006) 4782–4795.

[31] B. Ristic, S. Arulampalam, N. Gordon, Beyond the Kalman Filter, Artech House, 2004.

[32] S. Santini, B. Ostermaier, A. Vitaletti, First experiences using wireless sensor networks for noise pollution monitoring, in: Workshop on Real-World Wireless Sensor Networks REALWSN '08, ACM, 2008, pp. 61–65.

[33] Erik B. Sudderth, Alexander T. Ihler, Michael Isard, William T. Freeman, Alan S. Willsky, Nonparametric belief propagation, Commun. ACM 53 (October (10)) (2010) 95–103.