Research Article
An Advanced Auxiliary Delay-Weight Particle Filter with Linear Computation Cost

Chen Li, 1,2 Lin Sun, 1 Zengwei Zheng, 1 and Dan Chen 1

1 Hangzhou Key Laboratory for IoT Technology & Application, Zhejiang University City College, Hangzhou 310015, China
2 College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

Correspondence should be addressed to Zengwei Zheng; zhengzw@zucc.edu.cn

Received 9 March 2016; Revised 4 April 2016; Accepted 18 April 2016

Academic Editor: Federico Barrero

Copyright © 2016 Chen Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We investigate the problem of tracking mobile targets in wireless sensor networks. We propose an advanced auxiliary delayed-weight particle filter algorithm (ADWPF). We make a deep study on the evolvement of particles and formally define the tree-like structure relationship among particles based on observations. Most importantly, we add some auxiliary particles to these structures formed by sampled particles in order to obtain more efficient structures. Based on the newly tree-like structures formed by auxiliary particles and sampled particles, we design a well efficient delayed-weight algorithm with linear computation cost. Experiment results demonstrate that our algorithm can greatly improve the tracking accuracy of a mobile target, compared with bootstrap filter, auxiliary particle filter, and another delayed-weight particle filter.

1. Introduction

In recent years, tracking mobile targets in wireless sensor networks has attracted more and more attention of the researchers in academia and industry [1−3]. This technology has fairly good application prospects in indoor environments, such as shopping mall and underground parking.

Obviously, particle filter is one of the most important technologies for tracking mobile targets in wireless sensor networks. Many researchers in [4−6] have utilized particle filter to track the mobile targets in wireless sensor networks. Obviously, the performance of particle filter algorithm has a great impact on the tracking accuracy. So it is very meaningful and important to design an efficient particle filter algorithm. This is also the motivation of our research.

In our paper, we propose an auxiliary delayed-weight particle filter algorithm (ADWPF). Our main contributions are as follows:

(i) We make a deep study on the evolvement of particles and give the formal definition of the tree-like structure relationship among particles based on observations.

(ii) We add some auxiliary particles to the well-defined tree-like structures formed by sampled particles in order to get more efficient structures. These auxiliary particles can greatly help to improve tracking accuracy.

(iii) Based on the newly tree-like structures formed by auxiliary particles and sampled particles, we design a well efficient delayed-weight algorithm with linear computation cost, whose time complexity is $O(\Delta TN)$; here $\Delta$ is the delayed time length, $T$ is the number of tracking steps, and $N$ is the number of samples in each step.

The remainder of this paper is organized as follows. Section 2 reviews the related work in the area of particle filter and delayed-state filtering. In Section 3, we briefly overview the particle filter and describe the key technologies applied in our algorithm. In Section 4, we describe the deduction and the framework of our algorithm. Section 5 presents simulation results for a self-navigation problem in wireless sensor networks. Section 6 makes the conclusion.

2. Related Work

The particle filter algorithm has been widely used to infer the latent states of stochastic dynamic systems in [7].
It uses the sequential nature of stochastic dynamic systems to generate sequentially weighted Monte Carlo samples, and these samples are used to infer the latent states. For example, in the problem of self-navigation for a mobile target in wireless sensor networks, the position of the mobile target at each time step is the latent state to be inferred.

The bootstrap filter in [8] was the first successful application of the particle filter to the field of nonlinear filtering. It utilizes transition density function as the importance distribution to generate Monte Carlo samples of latent states. Then it assigns the weight of each sample based on the likelihood of the measurement. However, outliers in the measurement data will decrease the performance of the bootstrap filter as pointed in [8].

In order to improve the robustness of the sampling procedure and obtain approximate samples from the optimal importance distribution, which could minimize the variance of importance weights, [9] introduced the auxiliary particle filter. This algorithm can efficiently reduce the variance of samples’ weights.

Reference [10] pointed out that the “future” measurement information can be utilized to improve the estimation accuracy of current system state and mitigate degeneracy of the importance weights over time. So the approximations of the “earlier” marginal distributions of latent states will not collapse as time increases. Reference [10] introduced some delayed-sample methods, such as the Exact Lookahead Sampling method in [11] and the Block Sampling method in [12]. The key technology of these methods is to design an efficient importance distribution, which utilizes “future” information to generate weighted samples of the current states.

In fact, the delayed-sample algorithms which utilize “future” information to generate samples of current states are very similar to the smoothing algorithms which utilize current received measurement information to smooth the marginal filtering distributions of historical hidden states. Reference [10] also pointed that these two methods have very close relationship. Traditional smoothing algorithms mainly include the forward filtering-backward smoothing algorithm and the two-filter smoothing algorithm in [13]. The former is relying on the particle implementation of the forward filtering-backward smoothing formula to reweight the historical filtered samples, and the latter is based on the particle implementation of the two-filter smoothing formula to regenerate weighted samples for the historical hidden states. These two methods have time complexity $O(\Delta N^2)$ in [13], where $\Delta$ is the time interval and $N$ is the number of samples in each step.

3. Particle Filter Algorithm

3.1. System Description. Suppose that the dynamic system [14] is given by the following state and measurement equations:

$$x_{t+1} = f(x_t) + \omega_t,$$

$$y_{t+1} = g(x_{t+1}) + \nu_t,$$

where $x_t$ is state, $y_{t+1}$ is measurement, and $\omega_t, \nu_t \sim N(0, 1)$. Therefore, the transition probability function of state variable $x_{t+1}$ is $p(\cdot | x_t)$ and the likelihood function of measurement $y_{t+1}$ is $p(y_{t+1} | x_{t+1})$. The joint distribution of measurement information $y_{1:t}$ is $(y_1, \ldots, y_t)$ and state sequence $x_{1:t} = (x_1, \ldots, x_t)$ is

$$p(x_{1:t}, y_{1:t}) = \pi(x_1) \prod_{k=1}^{t} p_f(x_{k+1} | x_k) \prod_{k=1}^{t} p_g(y_k | x_k),$$

(3)

where $\pi(x_1)$ is the prior probability distribution of initial state. So the posterior distribution of $x_{1:t}$ is

$$p(x_{1:t} | y_{1:t}) = \frac{p(x_{1:t}, y_{1:t})}{p(y_{1:t})}. $$

(4)

3.2. Particle Filter. The aim of particle filter is to get the sequential Monte Carlo samples $X_{1:t} \sim p(x_{1:t} | y_{1:t})$, here $i = 1, 2, \ldots, N$, so the Monte Carlo approximation of $p(x_{1:t} | y_{1:t})$ is

$$\tilde{p}(x_{1:t} | y_{1:t}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{X_{1:t}}(x_{1:t}),$$

(5)

where $\delta(\cdot)$ is Delta function. The particle filter generally consists of the steps shown in Algorithm 1 (take bootstrap filter algorithm as an example).

The time complexity is $O(TN)$ at each time step, and memory complexity is $O(TN)$.

3.3. Delay Strategy. The key idea of this strategy is that the dynamic system offers more and more information of the latent states as time increases. In order to formalize this fact, assume that the measurement information available at time $t + \Delta$ takes the form $y_{1:t+\Delta} = (y_1, \ldots, y_{t+\Delta})$, the true system states are $x_{1:t+\Delta} = (x_1, \ldots, x_t)$, and $E(x_{1:t+\Delta} | y_{1:t+\Delta})$ is the expectation of $x_{1:t+\Delta}$ based on $y_{1:t+\Delta}$. Let $\tilde{X}_{1:t+\Delta}$ be a consistent Monte Carlo of $E(x_{1:t+\Delta} | y_{1:t+\Delta})$. The mean squared difference between $\tilde{X}_{1:t+\Delta}$ and $x_{1:t+\Delta}$ is $E((\tilde{X}_{1:t+\Delta} - x_{1:t+\Delta})^2)$ that follows.

**Proposition 1.** When the Monte Carlo sample size $N$ is sufficiently large, $E((\tilde{X}_{1:t+\Delta} - x_{1:t+\Delta})^2)$ decreases as $\Delta$ increases.

For the proof one can refer to [10].

Based on the statement of this proposition, we can design an algorithm that utilizes more and more information to improve the inference about the latent state in a specific time. This lays down the fundament of our work.

There are mainly two methods to implement the idea of Proposition 1 [10]. The first method is the delayed-sample method. It designed an importance distribution, which utilizes measurement information after time $t$ to regenerate weighted samples of state at time $t$, for example, auxiliary particle filter. The second method is the delayed-weight method, which utilized information after time $t$ to reweight samples $\{X_{1:t}\}_{t=1}^N$. 

International Journal of Distributed Sensor Networks
Step 1 (Initialization). At time $t = 1$, sample $\tilde{X}_1^i \sim \pi(x_1)$ to obtain the Monte Carlo approximation of $p(x_1 \mid y_1)$

$$\tilde{p}(x_1 \mid y_1) = \frac{1}{N} \sum_{i=1}^{N} W_i^1 \delta_{X_i^1}(x_1),$$

where $W_i^1 \propto p(y_1 \mid \tilde{X}_1^i)$ is the weight of $\tilde{X}_1^i$, resample $X_1^i \sim \tilde{p}(x_1 \mid y_1)$ to obtain

$$\tilde{p}(x_1 \mid y_1) = \frac{1}{N} \sum_{i=1}^{N} \delta_{X_i^1}(x_1).$$

Step 2 (Filtering). At time $t \geq 2$, sample $X_t^i \sim p(\cdot \mid X_{t-1}^i)$ to obtain the Monte Carlo approximation of $p(x_t \mid y_{1:t})$

$$\tilde{p}(x_t \mid y_{1:t}) = \frac{1}{N} \sum_{i=1}^{N} W_i^t \delta_{X_i^t}(x_t),$$

where $W_i^t \propto p(y_{1:t} \mid \tilde{X}_t^i)$ is the weight of $\tilde{X}_t^i$, resample $X_t^i \sim \tilde{p}(x_1 \mid y_{1:t})$ to obtain

$$\tilde{p}(x_t \mid y_{1:t}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{X_i^t}(x_t)$$

Algorithm 1: Bootstrap filter algorithm.

3.4. Relation between Delayed-Sample Method and Delayed-Weight Method

3.4.1. The Problem of Delayed-Sample Method. We know that the most important technology in particle filter is to choose an efficient importance distribution. The optimal importance distribution of the delayed-sample method has the following form [10]:

$$\tilde{p}(x_t \mid y_{1:t}) \propto \int p(X_{t-1}^i, x_t, y_{t+\Delta}) \, dx_{t-1} \cdots dx_{t+\Delta} \tag{6}$$

This optimal importance distribution can generate more effective samples than standard particle filter method, which is based on the one-at-a-time sampling strategy. For continuous state space, it is very hard to adopt this method, since it is not easy to get the explicit form of (6).

3.4.2. Delayed-Weight Method Is a Better Choice. In order to solve the problem of the delayed-sample method, we can discretize the continuous state space using sampled particles, which is the core idea of the delayed-weight method. Figure 1 illustrates the evolvement of particles in the standard particle filter method at time $t$.

Assume that we observe measurements $(y_1, \ldots, y_{t+\Delta})$ and have got weighted particles $\{X_{t-1}^i, w_{t-1}^i\}_{i=1}^{N}, \ldots, \{X_{t+\Delta}^i, w_{t+\Delta}^i\}_{i=1}^{N}$ using a forward particle filter method until time $t + \Delta$; then we can use these particles to represent the continuous state space, as illustrated in Figure 2.

The main advantage of doing so is that the optimal importance distribution $p(X_{t-1}^i, x_t, y_{t+\Delta})$ in (6) can be approximated by

$$p(X_{t-1}^i, x_t, y_{t+\Delta}) \approx \sum_{i_{t-1}=1}^{N} \cdots \sum_{i_{t+\Delta}=1}^{N} p(X_{t-1}^{i_{t-1}}, x_t, X_{t+\Delta}^{i_{t+\Delta}}, y_{t+\Delta}) \tag{7}$$

Therefore, the delayed weight $\bar{w}_t^i$ of particle $X_t^i$ satisfies

$$\bar{w}_t^i \propto p(X_{t-1}^i, X_t^i, y_{t+\Delta}) \approx \sum_{i_{t-1}=1}^{N} \cdots \sum_{i_{t+\Delta}=1}^{N} p(X_{t-1}^{i_{t-1}}, X_t^{i_{t}}, X_{t+\Delta}^{i_{t+\Delta}}, y_{t+\Delta}) \tag{8}$$

Obviously, this delayed weight can be calculated by the forward filtering-backward smoothing method. We name this method DWFB in [11, 15].

The main disadvantage of this method in [15] is that, in each iteration, we must make use of all particles at time $k + 1$.
Figure 2: Continuous state space can be discretized by weighted particles.

to calculate the delayed weight of each particle at time $k$; for example, the delayed weight $\hat{w}_k^i$ of $X_k^i$ is

$$\hat{w}_k^i \propto \sum_{j=1}^{N} \hat{w}_{k+1}^j P_j \left( X_{k+1}^j \mid X_k^i \right), \quad i = 1, \ldots, N. \quad (9)$$

Since the number of particles is $N$ at time $k$, the time complexity of this algorithm is $O(\Delta T N^3)$; when $N$ is large, the computation cost is very high. In order to reduce the time complexity, we propose an advanced delayed-weight particle filter weight method, which not only reduces the time complexity to $O(\Delta T N)$ but also greatly improves the tracking accuracy.

4. An Auxiliary Delayed-Weight Method

Our method consists of two key technologies: designing a backward smoothing procedure and adding auxiliary particles.

4.1. A Backward Smoothing Procedure

**Definition 2.** If one particle $X_t$ at time $t$ is sampled from the importance distribution $q(x_t \mid x_{t-1})$, where $X_{t-1}$ is a particle at time $t-1$ and $t = 1, \ldots, T$, one says that $X_t$ is the child of $X_{t-1}$, denoted by $X_t = \text{child}(X_{t-1})$ or, conversely, that $X_{t-1}$ is the parent of $X_t$, denoted by $X_{t-1} = \text{parent}(X_t)$. Furthermore, if $X_t$ has a child $X_{t+1}$, then $X_{t+1}$ is the grandchild of $X_{t-1}$, denoted by $X_{t+1} = \text{grandchild}(X_{t-1})$, and $X_{t-1}$ is the grandparent of $X_{t+1}$, denoted by $X_{t-1} = \text{grandparent}(X_{t+1})$.

Observing the evolution of particles in Figure 1, some particles have more than one child, while some have no child. Particularly, we observe that a particle and its child/grandchild form a tree-like structure, which is straightforward in Figure 2.

**Definition 3.** The tree that consists of particle $X$ and its children/grandchildren is called a generation tree, denoted by $\text{tree}(X)$. $X$ is the root of $\text{tree}(X)$.

As illustrated in Figure 3, the particles denoted by green circles construct the generation tree $\text{tree}(X^i_t)$ and $X^i_t$ is the root of $\text{tree}(X^i_t)$.

**Assumption 4.** If the particle $X_{t+4}$ is not the child/grandchild of $X_t$, then $p_j(X_{t+4} \mid X_t) = 0$, where $p_j(\cdot)$ is the transition probability function, $t = 1, \ldots, T$, and $k > 0$.

The main advantage of doing so is that we only need to make use of children/grandchildren of $X^i_t$ to calculate the delayed weight $\hat{w}_k^i$ of $X^i_k$, shown in Figure 4. Therefore, (8) can be reduced to

$$\hat{w}_k^i \propto p \left( X_{t-1}^i, X_t^i, y_{t+\Delta} \right) \quad (10)$$

where $N_{t+j}$ is the number of children/grandchildren of $X^i_t$ at time $t + j$, $j = 1, \ldots, \Delta$.

Observing the structure of $\text{tree}(X^i_t)$ in Figure 4(b), we design an advanced delayed-weight algorithm whose time complexity is $O(\Delta T N)$ to calculate $\hat{w}_k^i$. Take $\text{tree}(X^i_t)$ in Figure 4(b) as an example; here $\Delta = 3$. Since there are 5 leaf nodes in $\text{tree}(X^i)$, the delayed weight $\hat{w}_k^i$ of $X^i_k$ can be calculated as follows:

$$\hat{w}_k^i \propto p \left( X_{t-1}^i, X_{t+1}^{i+1}, X_{t+2}^{i+2}, X_{t+3}^{i+3}, y_{t+\Delta}; y_{t+\Delta+2} \right) + p \left( X_{t-1}^i, X_{t+1}^{i+1}, X_{t+2}^{i+2}, X_{t+3}^{i+3}, y_{t+\Delta} \right) + p \left( X_{t-1}^i, X_{t+2}^{i+2}, X_{t+3}^{i+3}, y_{t+\Delta+2} \right) + p \left( X_{t-1}^i, X_{t+2}^{i+2}, X_{t+3}^{i+3}, y_{t+\Delta} \right) \quad (11)$$

where $\{X_{t+1}^{i+1}, X_{t+2}^{i+2}, X_{t+3}^{i+3}, y_{t+\Delta}, y_{t+\Delta+2}\}$ are leaf nodes of $\text{tree}(X^i_t)$. Importantly, we find that every term in the right side of (11) includes one leaf node.

**Definition 5.** One calls the joint probability function representing the parent-child relation between leaf node $X_{\text{leaf}}$ and root node $X$ the backward smoothing term, denoted by $P(X_{\text{leaf}} \rightarrow X)$; for example, $P(X_{t+2}^{i+2} \rightarrow X_t^i) = p(X_t^i, X_{t+2}^{i+2}, X_{t+3}^{i+3})$.

Therefore $\hat{w}_k^i$ in (11) can be rewritten as

$$\hat{w}_k^i \propto P \left( X_{t+2}^{i+2} \rightarrow X_t^i \right) + P \left( X_{t+2}^{i+2} \rightarrow X_t^i \right) + P \left( X_{t+3}^{i+3} \rightarrow X_t^i \right) + P \left( X_{t+3}^{i+3} \rightarrow X_t^i \right) \quad (12)$$

More importantly, each backward smoothing term can be decomposed as the product of transition probability functions and likelihood functions; for example, the backward
smoothing term \( P(X_{i+1}^{t+1} \rightarrow X_i^t) \) can be decomposed as follows:

\[
P(X_{i+1}^{t+1} \rightarrow X_i^t) = P(X_i^t, X_{i+1}^{t+1}, y_{t+2})
= p(X_i^t) p_f(X_{i+1}^{t+1} | X_i^t) p_f(y_{t+2} | X_{i+1}^{t+1})
\]

\[
\times p_g(y_{t+1} | X_{i+2}^{t+1}) p_g(y_{t+2} | X_{i+3}^{t+1}).
\]

(13)

Other backward smoothing terms can be decomposed in the same way. Moreover, from (13), we can see that it is very straightforward to use an iterate method to calculate each backward smoothing term, which is the core idea of our backward smoothing algorithm. Procedure 1 gives the backward smoothing (BS) process from one leaf node to the root node.

Obviously, the time complexity of Procedure 1 is \( O(\Delta) \). Figure 5 shows the backward smoothing procedure.

4.2. Adding Auxiliary Particles. Experiment results demonstrate that it is not enough to use particles in tree(\( X_i^t \)) to calculate the delayed weight \( \hat{w}_i^k \) of \( X_i^k \). We should add some auxiliary particles to tree(\( X_i^t \)) as illustrated in Figure 6.

**Definition 6.** One calls the modified generation tree to which auxiliary particles have been added the auxiliary generation tree, denoted by aux\_tree(\( X \)), where \( X \) is the root of aux\_tree(\( X \)).

In Figure 6, we also call these newly added auxiliary particles, like \( u_{i+2}^1, u_{i+3}^1 \), and \( u_{i+3}^2, \) grandchildren of \( X_i^t \). The most important feature of aux\_tree(\( X_i^t \)) is that all of its leaf nodes have the same depth, which equals the delayed time length \( \Delta \).

**Definition 7.** The state of auxiliary particle is defined as the expectation of transition probability function; that is, the auxiliary particles \( u_{i+2}^1 \) and \( u_{i+3}^1 \) in Figure 6 are defined as follows:

\[
u_{i+2}^1 = E_{P_f}(\cdot | X_{i+2}^{t+1}),
\]

\[
u_{i+3}^1 = E_{P_f}(\cdot | u_{i+2}^1).
\]

(14)
Input: a leaf node $X_i^t$; Output: the backward smoothing term $P(X_i^t \rightarrow X_{\text{root}})$, where $X_{\text{root}}$ is the grandparent of $X_i^t$ at time $t-\Delta$.

(1) Initialization: $P(X_i^t \rightarrow X_{\text{root}}) = 1$;
(2) $X_{\text{temp}} = X_i^t$;
(3) for $k = t, \ldots, t-\Delta+1$
   // the data structure of $X_i^k$ includes a pointer to his parent
   (4) $X_{\text{parent}} = \text{parent}(X_{\text{temp}}); //a backward smoothing
   $//operation$
   (5) $P(X_i^k \rightarrow X_{\text{parent}}) = P(X_i^t \rightarrow X_{\text{root}}) \ast p(y_k | X_{\text{temp}}) \ast p_j(X_{\text{temp}} | X_{\text{parent}})$;
   (6) $X_{\text{temp}} = X_{\text{parent}}$;
   end
(7) return $P(X_i^t \rightarrow X_{\text{root}})$;

Figure 5: The backward smoothing procedure of Procedure 1. The calculation of each backward smoothing term is iterated from one leaf node to the root node.

Furthermore, according to the definition of transition probability function (1), we have

\[
\begin{align*}
    u_{t+2}^1 &= f(X_{t+2}^i) + E(w_i) = f(X_{t+2}^i), \\
    u_{t+3}^1 &= f(u_{t+2}^i) + E(w_i) = f(u_{t+2}^i),
\end{align*}
\]

(15)

where since $w_i \sim \mathcal{N}(0, 1), E(w_i) = 0$.

According to aux_tree($X_i^t$) in Figure 6, the delayed weight $\tilde{w}_k^i$ of $X_k^i$ can be calculated as follows:

\[
\begin{align*}
    \tilde{w}_k^i \propto P(X_i^t, X_{t+1}^i, X_{t+2}^i, u_{t+2}^i, Y_{t+2}^i) \\
    + P(X_i^t, X_{t+1}^i, X_{t+2}^i, u_{t+3}^i, Y_{t+3}^i) \\
    + P(X_i^t, X_{t+1}^i, X_{t+2}^i, u_{t+3}^i, Y_{t+3}^i) \\
    + P(X_i^t, X_{t+1}^i, X_{t+2}^i, u_{t+3}^i, Y_{t+3}^i) \\
    + P(X_i^t, X_{t+1}^i, X_{t+2}^i, u_{t+3}^i, Y_{t+3}^i),
\end{align*}
\]

(16)

where $\{u_{t+2}^i, X_{t+1}^i, X_{t+2}^i, X_{t+3}^i, u_{t+3}^i, X_{t+3}^i\}$ are leaf nodes of aux_tree($X_i^t$). It is a little different from (11). And, according to the definition of backward smoothing term, $\tilde{w}_k^i$ can also be rewritten as

\[
\begin{align*}
    \tilde{w}_k^i \propto P(u_{t+3}^i \rightarrow X_i^t) + P(X_{t+1}^i \rightarrow X_i^t) \\
    + P(X_{t+2}^i \rightarrow X_i^t) + P(u_{t+3}^i \rightarrow X_i^t) \\
    + P(X_{t+3}^i \rightarrow X_i^t).
\end{align*}
\]

(17)

Absolutely, each backward smoothing term in (17) can be calculated by Procedure 1.

The main advantage of adding auxiliary particles to the generation tree is that they can greatly improve the filtering accuracy, which is verified in our experiments.

Procedure 2 gives the process of constructing an auxiliary generation tree (CAGF).

Lemma 8. The number of leaf nodes in $\text{tree}(X_i^t)$ is equal to the number of leaf nodes in aux_tree($X_i^t$).

Proof. According to step (13) in Procedure 2, only one child is added to every leaf node. So the number of leaf nodes will not increase. \(\square\)

Lemma 9. Let $N_k$ denote the number of children/grandchildren of $X_i^t$ in $\text{tree}(X_i^t)$ at time $k$, where $t+1 \leq k \leq t+\Delta$. The time complexity $T_{\text{CAGF}}(X_i^t)$ of Procedure 2 satisfies

\[
\mathcal{O}\left(\sum_{k=t+1}^{t+\Delta} N_k\right) \leq T_{\text{CAGF}}(X_i^t) \leq \mathcal{O}\left((\Delta+1) \sum_{k=t+1}^{t+\Delta} N_k\right).
\]

(18)

The proof is obvious.

4.3. Algorithm Framework. In this section, we first give the framework of our algorithm, and then we analyze its time complexity.

Theorem 10. The time complexity $T_{\text{ADWPF}}$ of Algorithm 3 is $O(CN), C \ll N$.

Proof. Obviously, the time cost of Algorithm 3 mainly consists of steps (4) and (6). Let $T_{\text{CAGF}}(X_{t-\Delta}^i), 1 \leq i \leq N,$
Let $\Delta$ be the height of tree($X_i^t$), $N_k$ is the number of child/grandchildren of $X_i^t$ at time $k$, where $t + 1 \leq k \leq t + \Delta$.

**Input:** tree($X_i^t$)

**Output:** an auxiliary generation tree aux_tree($X_i^t$)

(1) Initialization: index = 0; //index auxiliary particles
(2) if $X_i^t$ has no child
(3) $X = X_i^t$;
(4) for $j = k + 1, \ldots, t + \Delta$
(5) $u_j^{\text{index}} = E_{p_j} (\cdot | X)$;
(6) $X = u_j^{\text{index}}$;
(7) index = index + 1;
end
end
(8) else for $k = t + 1, \ldots, t + \Delta - 1$
(9) for $n = 1, \ldots, N_k$
(10) if $X_n^t$ is a leaf node
(11) $X = X_n^t$;
//add auxiliary particles
(12) for $j = k + 1, \ldots, t + \Delta$
(13) $u_j^{\text{index}} = E_{p_j} (\cdot | X)$;
(14) $X = u_j^{\text{index}}$;
(15) index = index + 1;
end
end
end
end
(16) return aux_tree($X_i^t$)

**Procedure 2: CAGF.**

\[
\sum_{i=1}^{N} T_{CAGF}(X_i^{t-\Delta}) \leq \sum_{i=1}^{N} \Theta \left( (\Delta + 1) \sum_{k=t-\Delta+1}^{t} N_k^i \right) = \Theta \left( (\Delta + 1) N \right),
\]
where the second equation is based on the fact that we only sampled \( N \) particles at each time step according to step (1) in ADWPF. And according to Lemma 8

\[
\begin{align*}
\sum_{i=1}^{N} T_{CAGF} \left( X_{t-\Delta}^{i} \right) & \geq \sum_{i=1}^{N} \Theta \left( \sum_{k=1}^{t} N_{k}^{i} \right) \\
& = \Theta \left( \sum_{i=1}^{N} \sum_{k=1}^{t} N_{k}^{i} \right) = \Theta \left( N \right).
\end{align*}
\]

Let \( T_{BS} \left( X_{t-\Delta}^{i} \right) \), \( 1 \leq i \leq N \), denote the time cost of step (6) in ADWPF. The sum number of leaf nodes of all generation trees \( \{ \text{tree}(X_{t-\Delta}^{1}), \ldots, \text{tree}(X_{t-\Delta}^{N}) \} \) is less than \( \Delta N \) and greater than \( N \), so the sum number of leaf nodes of all auxiliary generation trees \( \{ \text{aux}_\text{tree}(X_{t-\Delta}^{1}), \ldots, \text{aux}_\text{tree}(X_{t-\Delta}^{N}) \} \) is less than \( \Delta N \) and greater than \( N \) according to Lemma 8. Since the time cost of Procedure 1 is \( O(\Delta) \) for each leaf node, we have

\[
N \ast O(\Delta) \leq \sum_{i=1}^{N} T_{BS} \left( X_{t-\Delta}^{i} \right) \leq \Delta N \ast O(\Delta)
\]

\[
= O(\Delta^2 N). \tag{21}
\]

So for the time complexity \( T_{ADWPF} \) of Algorithm 2 we have

\[
T_{ADWPF} = T \ast \left( \sum_{i=1}^{N} T_{CAGF} \left( X_{t-\Delta}^{i} \right) \right) + \sum_{i=1}^{N} T_{BS} \left( X_{t-\Delta}^{i} \right)
\]

\[
\leq T \ast \left( \Theta \left( (\Delta + 1) N \right) \right) + O \left( \Delta^2 N \right)
\]

\[
= \Theta \left( (\Delta + 1) N \right). \tag{22}
\]

Meanwhile,

\[
T_{ADWPF} = T \ast \left( \sum_{i=1}^{N} T_{CAGF} \left( X_{t-\Delta}^{i} \right) \right) + \sum_{i=1}^{N} T_{BS} \left( X_{t-\Delta}^{i} \right)
\]

\[
\geq T \ast \Theta \left( N \right) + O \left( \Delta N \right) \tag{23}
\]

\[
= \Theta \left( (\Delta + 1) N \right),
\]

where \( \Delta, T \ll N \). So the time complexity of ADWPF is \( O(\Delta N) \), \( C \ll N \).

\( \Box \)

5. Simulation and Results

To verify the performance of our algorithm ADWPF, we consider a wireless sensor network with 70 anchor sensors \( \{ a_{i} = (a_{ix}^{i}, a_{iy}^{i}) \}_{i=1}^{70} \). These sensors are randomly distributed in a 50 x 50 deployment area. Our task is to track a mobile sensor that moves within the area. The state of the mobile sensor at time \( t \) is \( x_{t} = (x_{t}, y_{t}, \theta_{t})^T \), where \( (x_{t}, y_{t}) \) is the true location of the mobile sensor and \( \theta_{t} \) is its velocity. In our simulation, we set \( (x_{t}, y_{t})^T = (1, 1)^T \). So we just need to utilize our algorithm to infer the state \( \tilde{x}_{t} = (\tilde{x}_{t}, \tilde{y}_{t}, \tilde{\theta}_{t})^T \) of the mobile sensor at time \( t \). The communication radius of sensor is set as 5. At time \( t \), the distance measurement \( y_{t} \) from anchor sensor \( a_{i} \) is

\[
y_{t} = \sqrt{(a_{ix}^{i} - \tilde{x}_{t})^2 + (a_{iy}^{i} - \tilde{y}_{t})^2} + n, \tag{24}
\]

where the distance measurement noise \( n \in R \) follows \( n \sim \mathcal{N}(0, 1) \).

5.1. Linear Motion Model. The state of the mobile sensor evolves according to [16, 17]:

\[
x_{t} = \Phi x_{t-1} + \Gamma w_{t}, \tag{25}
\]

where \( w_{t} \in R^2, w_{t} \sim \mathcal{N}(\begin{bmatrix} 0 \ 0 \ 0 \ 0 \end{bmatrix}^T, \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix}) \), and \( \Phi \) and \( \Gamma \) have the following form:

\[
\Phi = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \Gamma = \begin{bmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{bmatrix}. \tag{26}
\]

The initial state of the mobile sensor is \( x_{0} = (0, 1, 0, 1)^T \), the sampling interval is \( y = 1 \), the number of tracking steps is \( T = 50 \), the number of simulations is \( n = 50 \), and we use the mean squared error (MSE) as the performance measurement. It is defined as follows:

\[
\text{MSE} = \frac{1}{T \ast \text{num}} \sum_{j=1}^{n} \sum_{t=1}^{T} (\tilde{x} - \text{est}_{\tilde{x}})^2, \tag{27}
\]

where \( \tilde{x} \) is the true location of the mobile sensor at time \( t \) in the \( jth \) simulation and \( \text{est}_{\tilde{x}} \) is the estimation of the location.

Table 1 compares the average MSE for bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in linear motion model.

5.2. Gaussian-Markov Motion Model. In this test, the target node is moving according to the Gaussian-Markov motion model [18]. In this model, we can use one tuning parameter to vary the degree of randomness of the movement, and the target node will not move out of the deployment area [19]. The speed \( v_{x} \) and direction \( d_{x} \) of the mobile target evolves as follows:

\[
v_{x} = \alpha v_{x-1} + (1 - \alpha) v + \sqrt{(1 - \alpha^2)n_{v}},
\]

\[
d_{x} = \alpha d_{x-1} + (1 - \alpha) d + \sqrt{(1 - \alpha^2)n_{d}}, \tag{28}
\]

where \( n_{v} \) and \( n_{d} \) are white noise, \( \alpha \) is the tuning parameter used to control the randomness, and \( v \) and \( d \) are constants.
Assume $N$ is the number of particles, $t$ is current time index, count is the number of simulation steps, $\Delta$ is the delayed time interval, $w_i^t$ is the filtered weight and $\hat{w}_i^t$ is the modified weight.

(1) for $t = 1, \ldots, T$
(2) Generate samples $\{X_i^t, w_i^t, \hat{w}_i^t = 0\}_{i=1}^N$ through a forward particle filter (like Algorithm 1).
(3) if $t > \Delta$
(4) for $i = 1 : N$
(5) $\hat{w}_i^t = w_i^t$
(6) for $k = T - 1 : T - \Delta$
(7) for $i = 1 : N$
\[
\hat{w}_i^t = \frac{w_i^t P_j^t (X_i^{k+1} \mid X_i^k)}{\sum_{j=1}^N w_j^t P_j^t (X_i^{k+1} \mid X_i^k)}
\]
end
end
end
end

The inference result of system state $x_{T-\Delta}$ is
\[
\hat{x}_{T-\Delta} = \sum_{i=1}^N \hat{w}_i^{T-\Delta} \cdot X_i^{T-\Delta}
\]

**Algorithm 2: DWFB.**

Let $N$ is the number of particles, $t$ is current time index, $\Delta$ is the delayed time length, $w_i^t$ is the forward filtered weight and $\hat{w}_i^t$ is the delayed weight.

(1) for $t = 1, \ldots, T$
   Generate samples $\{X_i^t, w_i^t, \hat{w}_i^t = 0\}_{i=1}^N$ through a forward particle filter (like Algorithm 1).
end
(2) if $t > \Delta$
(3) for $i = 1, \ldots, N$
   //Procedure 2.
(4) aux_tree$(X_i^{t-\Delta}) = CAGF$(tree$(X_i^{t-\Delta})$, $\Delta$)
(5) for each leaf node $X_j^i$ of aux_tree$(X_i^{t-\Delta})$
   //Procedure 1.
(6) $P(X_j^i \rightarrow X_i^{t-\Delta}) = BS(X_j^i)$;
(7) $\hat{w}_i^{t-\Delta} = \hat{w}_i^{t-\Delta} + P(X_j^i \rightarrow X_i^{t-\Delta})$;
(8) $X_i = X_i^{t-\Delta}$;
end
end

**Algorithm 3: ADWPF.**

| MSE          | $N = 50$ | $N = 100$ | $N = 200$ | $N = 500$ | $N = 800$ |
|--------------|---------|---------|---------|---------|---------|
| Bootstrap particle filter | 0.129 | 0.081 | 0.055 | 0.042 | 0.039 |
| Auxiliary particle filter | 0.112 | 0.071 | 0.053 | 0.040 | 0.040 |
| DWFB ($\Delta = 1$) | 0.123 | 0.070 | 0.055 | 0.042 | 0.039 |
| DWFB ($\Delta = 3$) | 0.107 | 0.065 | 0.044 | 0.030 | 0.029 |
| DWFB ($\Delta = 5$) | 0.106 | 0.058 | 0.034 | 0.025 | 0.024 |
| ADWPF ($\Delta = 1$) | 0.117 | 0.063 | 0.047 | 0.035 | 0.032 |
| ADWPF ($\Delta = 3$) | **0.097** | 0.053 | 0.034 | 0.019 | 0.017 |
| ADWPF ($\Delta = 5$) | 0.098 | **0.049** | **0.024** | **0.015** | **0.012** |
Table 2: Average MSE for bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in Gaussian-Markov motion model.

| MSE                | N = 50  | N = 100 | N = 200 | N = 500 | N = 800 |
|--------------------|---------|---------|---------|---------|---------|
| Bootstrap particle filter | 2.189   | 2.038   | 1.908   | 1.870   | 1.859   |
| Auxiliary particle filter   | 2.194   | 2.063   | 1.993   | 2.021   | 2.028   |
| DWFB (Δ = 1)            | 2.577   | 2.254   | 2.159   | 2.103   | 2.101   |
| DWFB (Δ = 3)            | 2.598   | 2.399   | 2.399   | 2.312   | 2.344   |
| DWFB (Δ = 5)            | 2.029   | 1.911   | 1.781   | 1.763   | 1.766   |
| ADWPF (Δ = 1)           | 1.880   | 1.546   | 1.441   | 1.370   | 1.366   |
| ADWPF (Δ = 3)           | 1.204   | 0.999   | 0.880   | 0.795   | 0.781   |
| ADWPF (Δ = 5)           | 1.178   | 0.893   | 0.748   | 0.606   | 0.603   |

Table 3: Total time (sec.) cost of bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in linear motion model (T = 50).

| MSE                | N = 50  | N = 100 | N = 200 | N = 500 | N = 800 |
|--------------------|---------|---------|---------|---------|---------|
| Bootstrap particle filter | 0.017   | 0.028   | 0.056   | 0.141   | 0.230   |
| Auxiliary particle filter   | 0.031   | 0.062   | 0.121   | 0.301   | 0.497   |
| DWFB (Δ = 1)            | 0.246   | 0.927   | 3.713   | 22.322  | 56.934  |
| DWFB (Δ = 3)            | 0.663   | 2.592   | 10.494  | 63.828  | 164.757 |
| DWFB (Δ = 5)            | 1.040   | 4.096   | 16.320  | 101.648 | 259.860 |
| ADWPF (Δ = 1)           | 0.042   | 0.066   | 0.124   | 0.312   | 0.512   |
| ADWPF (Δ = 3)           | 0.084   | 0.153   | 0.317   | 0.803   | 1.300   |
| ADWPF (Δ = 5)           | 0.154   | 0.308   | 0.610   | 1.520   | 2.458   |

Table 4: Total time (sec.) cost of bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in Gaussian-Markov motion model (T = 50).

| MSE                | N = 50  | N = 100 | N = 200 | N = 500 | N = 800 |
|--------------------|---------|---------|---------|---------|---------|
| Bootstrap particle filter | 0.011   | 0.021   | 0.037   | 0.101   | 0.161   |
| Auxiliary particle filter   | 0.021   | 0.038   | 0.069   | 0.184   | 0.301   |
| DWFB (Δ = 1)            | 0.271   | 0.925   | 3.654   | 22.719  | 58.071  |
| DWFB (Δ = 3)            | 0.679   | 2.632   | 10.448  | 65.003  | 166.134 |
| DWFB (Δ = 5)            | 1.051   | 4.167   | 16.664  | 103.563 | 263.259 |
| ADWPF (Δ = 1)           | 0.044   | 0.055   | 0.109   | 0.266   | 0.444   |
| ADWPF (Δ = 3)           | 0.087   | 0.147   | 0.287   | 0.709   | 1.135   |
| ADWPF (Δ = 5)           | 0.142   | 0.276   | 0.547   | 1.764   | 2.179   |

representing the mean value of speed and direction. The location \( \text{loc}_t = (\text{loc}_{x,t}, \text{loc}_{y,t})^T \) of the mobile target evolves as follows:

\[
\begin{align*}
\text{loc}_{x,t} &= \text{loc}_{x,t-1} + v_t \cos d_t, \\
\text{loc}_{y,t} &= \text{loc}_{y,t-1} + v_t \sin d_t.
\end{align*}
\] (29)

We set \( \alpha = 0.75, v = 1, d = 90^\circ, \text{loc}_{x,0} = 25, \text{loc}_{y,0} = 25, \) and \( \gamma_0 = 1. \) Table 2 compares the average MSE for bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in Gaussian-Markov motion model.

As illustrated in Tables 1 and 2, with \( N \) fixed, the MSE values of our method ADWPF with different \( \Delta \) are all smaller than those of bootstrap particle filter and auxiliary particle filter. Furthermore, with \( N \) and \( \Delta \) fixed, the MSE value of ADWPF is smaller than that of DWFB. Tables 3 and 4 compare the total time cost of bootstrap particle filter, auxiliary particle filter, DWFB, and our method ADWPF in linear motion model \((T = 50)\) and Gaussian-Markov motion model \((T = 50)\). Since our method ADWPF takes the delay strategy, its time cost is a little higher than bootstrap particle filter and auxiliary particle filter, but our method efficiently utilizes the extra time cost to improve tracking accuracy, which is verified in Tables 1 and 2. Comparing to the delay-weight method DWFB in [15], our method ADWPF takes full advantage of the structure relationship among particles, so the time cost of ADWPF is far less than that of DWFB. These experiment results demonstrate the excellent performance of our algorithm.

6. Conclusion

Particle filter is a very important technology for tracking mobile targets in wireless sensor networks. In order to
improve the tracking accuracy of the mobile node, we propose an advanced auxiliary delay-weight particle filter algorithm. By adding some auxiliary particles to the generation trees formed by sampled particles, our algorithm ADWPF can efficiently utilize measurement information after time $t$ to reweight Monte Carlo samples $\{X_i^t\}_{i=1}^N$ generated at time $t$ without creating a substantially high computation burden. Experiment results demonstrate that our algorithm ADWPF can greatly improve the tracking accuracy of a mobile node, compared with bootstrap filter, auxiliary particle filter, and another delayed-weight particle filter in [15].

**Competing Interests**

The authors declare that there are no competing interests regarding publication of this paper.

**Acknowledgments**

This work has been supported by Hangzhou Key Laboratory for IoT Technology & Application.

**References**

[1] R. Harle, “A survey of indoor inertial positioning systems for pedestrians,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[2] S. Gezici, Z. Tian, G. B. Giannakis et al., “Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks,” *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 70–84, 2005.

[3] H. Shin, Y. Chon, and H. Cha, “Unsupervised construction of an indoor floor plan using a smartphone,” *IEEE Transactions on Systems, Man & Cybernetics Part C: Applications and Reviews*, vol. 42, no. 6, pp. 889–898, 2012.

[4] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao, “Travi-Navi: self-deployable indoor navigation system,” in *Proceedings of the 20th ACM Annual International Conference on Mobile Computing and Networking (MobiCom ’14)*, pp. 471–482, ACM, Maui, Hawaii, USA, September 2014.

[5] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, “Zee: zero-effort crowdsourcing for indoor localization,” in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom ’12)*, pp. 293–304, 2012.

[6] P.-H. Tseng, Z. Ding, and K.-T. Feng, “Cooperative self-navigation in a mixed LOS and NLOS environment,” *IEEE Transactions on Mobile Computing*, vol. 13, no. 2, pp. 350–363, 2014.

[7] H. R. Künsch, “Particle filters,” *Bernoulli*, vol. 19, no. 4, pp. 1391–1403, 2013.

[8] O. Cappe, S. J. Godsill, and E. Moulines, “An overview of existing methods and recent advances in sequential Monte Carlo,” *Proceedings of the IEEE*, vol. 95, no. 5, pp. 899–924, 2007.

[9] M. K. Pitt and N. Shephard, “Filtering via simulation: auxiliary particle filters,” *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599, 1999.

[10] M. Lin, R. Chen, and J. S. Liu, “Lookahead strategies for sequential Monte Carlo,” *Statistical Science*, vol. 28, no. 1, pp. 69–94, 2013.

[11] R. Chen, X. Wang, and J. S. Liu, “Adaptive joint detection and decoding in flat-fading channels via mixture Kalman filtering,” *IEEE Transactions on Information Theory*, vol. 46, no. 6, pp. 2079–2094, 2000.

[12] A. Doucet, M. Briers, and S. Sénécal, “Efficient block sampling strategies for sequential Monte Carlo methods,” *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 693–711, 2006.

[13] A. Doucet and A. M. Johansen, “A tutorial on particle filtering and smoothing: fifteen years later,” *Handbook of Nonlinear Filtering*, vol. 12, no. 4, pp. 656–704, 2009.

[14] O. Straka, J. Dunik, M. Simandl, and E. Blasch, “Comparison of adaptive and randomized unscented Kalman filter algorithms,” in *Proceedings of the IEEE 17th International Conference on Information Fusion (FUSION ’14)*, pp. 1–8, Salamanca, Spain, July 2014.

[15] M. Shan, S. Worrall, and E. Nebot, “Delayed-state nonparametric filtering in cooperative tracking,” *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 962–977, 2015.

[16] F. Meyer, O. Hlinka, and F. Hlawatsch, “Sigma point belief propagation,” *IEEE Signal Processing Letters*, vol. 21, no. 2, pp. 145–149, 2014.

[17] J. H. Kotecha and P. M. Djuric, “Gaussian particle filtering,” *IEEE Transactions on Signal Processing*, vol. 51, no. 10, pp. 2592–2601, 2003.

[18] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[19] V. Savic and S. Zazo, “Cooperative localization in mobile networks using nonparametric variants of belief propagation,” *Ad Hoc Networks*, vol. 11, no. 1, pp. 138–150, 2013.
