Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-00

M.Sc. Thesis

Distributed Particle Filtering

Rui Tang B.Sc.

Abstract

Extensive research has been conducted in the literature for multi-agent networks due to the advances in wireless communication, microfabrication, and integration. In many applications of multi-agent networks, the physical systems consist of massive nonlinear and non-Gaussian elements. Hence, in the first decade of this century, intensive research on distributed particle filters (DPFs) has been conducted to address the distributed estimation problems. For distributed algorithms, communication overhead is an important metric in terms of engineering feasibility. In previous work, the approach to distributed particle filtering relies on a parameterization of the posterior probability or likelihood function, to reduce communication requirements. However, as more and more effective resampling algorithms are proposed, the dependence of particle filter performance on particle set size is greatly reduced, so this thesis attempts to explore the possibility of DPFs based on direct particle exchange. In this article, the Gaussian process enhanced resampling algorithm is used. Meanwhile, several metaheuristic optimization algorithms (i.e., genetic algorithm and firefly algorithm) are further adapted to seek the global optimal particle set to improve the estimation performance. Furthermore, all algorithms are simulated in target tracking scenarios and are evaluated from three aspects: time, space, and communication complexity.

Distributed Particle Filtering

Rui Tang B.Sc.

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Microelectronics & Computer Engineering
Circuits and Systems Group

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Distributed Particle Filtering"** by **Rui Tang B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 20 July 2022

Chairman: _____
dr. R.T. Rajan

Advisor: _____
dr. R.T. Rajan

Committee Members: _____
dr.ir. J. Dauwels

_____
dr. F. Fioranelli

Abstract

Extensive research has been conducted in the literature for multi-agent networks due to the advances in wireless communication, microfabrication, and integration. In many applications of multi-agent networks, the physical systems consist of massive nonlinear and non-Gaussian elements. Hence, in the first decade of this century, intensive research on distributed particle filters (DPFs) has been conducted to address the distributed estimation problems. For distributed algorithms, communication overhead is an important metric in terms of engineering feasibility. In previous work, the approach to distributed particle filtering relies on a parameterization of the posterior probability or likelihood function, to reduce communication requirements. However, as more and more effective resampling algorithms are proposed, the dependence of particle filter performance on particle set size is greatly reduced, so this thesis attempts to explore the possibility of DPFs based on direct particle exchange. In this article, the Gaussian process enhanced resampling algorithm is used. Meanwhile, several metaheuristic optimization algorithms (i.e., genetic algorithm and firefly algorithm) are further adapted to seek the global optimal particle set to improve the estimation performance. Furthermore, all algorithms are simulated in target tracking scenarios and are evaluated from three aspects: time, space, and communication complexity.

Acknowledgments

The process of completing the dissertation research is undoubtedly challenging and meaningful. Here, I would like to express my sincere appreciation to all the people who have helped me, as well as my efforts and persistence along the way.

First of all, I would like to express my sincere thanks to my supervisor, dr. R.T. Rajan, who provided many valuable suggestions for the research and writing of my thesis. In addition, I would also like to thank doctoral student Ir. Ellen Riemens for her help in writing and revising my thesis. The weekly routine has been very helpful and enjoyable for me.

I would also like to thank the entire CAS group for providing me with the opportunity to study and research. Relevant staffs always patiently gave me guidance and help whenever I had difficulties.

Finally, I would like to thank all my family and friends for their generosity and invaluable help during these short two years of graduate study.

Rui Tang B.Sc.
Delft, The Netherlands
20 July 2022

Contents

List of Figures

List of Tables

Notation

Symbol	Description
m	particle index
n	time index
k	agent index
d	dimensions of the state space
$\|E\|$	the number of links in the multi-agent network
$\boldsymbol{x}_n$	target state at the nth time instant
$\boldsymbol{x}_n^m$	the mth particle at the nth time instant
$\boldsymbol{x}_n^{m,k}$	the mth particle of agent k at the nth time instant
$\boldsymbol{\omega}_n^m$	weight of particle m at the nth time instant
$\boldsymbol{\omega}_n^{m,k}$	weight of particle m of agent k at the nth time instant
$\boldsymbol{y}_n$	overall measurements at the nth time instant
$\boldsymbol{y}_{n,k}$	measurement taken at nth time instant by agent k
$\boldsymbol{g}_n(\cdot)$	state transition function at the nth time instant
$\boldsymbol{h}_{n,k}(\cdot)$	measurement function of agent k at the nth time instant
$\boldsymbol{u}_n$	process noise at the nth time instant
$\boldsymbol{v}_{n,k}$	measurement noise of agent k at the nth time instant
z	the number of reserved Laplacian transform coefficients in GLDPF
N_{knn}	the number of nearest neighbors in k-NN algorithm
C	the number of components in Gaussian mixture model
D	degree matrix
$\epsilon_{k,j}$	Metropolis weight between agent k and agent j
$\varepsilon/\boldsymbol{\varepsilon}$	noise(scalar/vector)
M	the number of particles
K	the number of agents
T	time length
L	the number of iterations(under different conditions, there will be different subscripts)
$(\cdot)^T$	transpose operator
$(\cdot)^{-1}$	inverse operator
$\lceil\cdot\rceil/\lfloor\cdot\rfloor$	ceiling function/floor function
$\|\cdot\|_2$	Euclidean norm
$\mathcal{N}(\cdot)$	normal distribution
$\mathcal{U}_{[a,b]}$	uniform distribution between a and b
$\mathbb{E}(\cdot)$	Expectation operator
$\mathcal{O}(\cdot)$	complexity operator
$\mathcal{GP}(\cdot)$	Gaussian Process(GP) model
$\kappa(\cdot)$	covariance between points in GP model(refer to (3.7))
$\mathcal{K}(\cdot)$	covariance matrix in GP model(refer to (3.7))

Note:

(1) The subscript n of function $\boldsymbol{g}(\cdot)$ and $\boldsymbol{h}(\cdot)$ is usually omitted in this article because the model we use is time independent.

(2) When the subscript k is missing, we default to only one agent.

1 Introduction

1.1 Background

Extensive research has been conducted in the literature for multi-agent networks due to the advances in wireless communication, micro-fabrication, and integration. In a multi-agent network, different agents are allowed to cooperate to solve a given problem and come up with solutions beyond the individual knowledge of each agent. A multi-agent network usually has the following listed advantages over a single agent network.

- The computational burden of solving a given complex problem can be distributed to multiple agents to reduce the computational resource requirements of a single agent. It also means that the agent does not have to pursue high-performance processors.
- Multi-agent networks address the situations where information from sources that are spatially distributed.
- Multi-agent networks improve overall system performance, especially in terms of scalability, robustness, flexibility, etc. For example, by distributing computing tasks to multiple agents, the system can be protected from the "single point of failure" problem typically associated with centralized systems.

Some typical examples of multi-agent networks includes wireless sensor networks [1], robot networks [2], unmanned aerial vehicles(UAVs) networks [3], satellite swarms [4]. Possible application scenarios include but are not limited to:

- Target tracking [1]
  In many practical applications such as military battlefield awareness and warehouse management, locating and tracking moving objects through various measurements such as bearings-angle gathered by sensor networks are the essential capabilities. In the military, for example, by rapidly deploying wireless sensor networks, applying efficient target tracking algorithms can gather information about the presence of enemy targets and track their actions on the battlefield. The target tracking problem presents the most important issues related to collaborative processing, information sharing, and group management, including the communication frequency, which sensors should sense at a given time, which sensors have useful information and should communicate.
- Environmental monitoring [1]
  The sensor can be used to monitor the survival of wildlife in special areas such as national nature reserves. Sensors can also monitor air quality and track environmental pollutants, providing scientific guidance for follow-up measures. In

addition, sensors can monitor chemical plumes to provide early warning. Seismic monitoring is another application area. Environmental monitoring is one of the earliest applications of sensor networks. An important consideration is the long-term durability of the sensor in an unattended environment.

- Surveillance [5]
  surveillance systems assist the human security personnel to identify abnormal conditions by performing object detecting and tracking tasks. Information from heterogeneous cameras (such as vision, infrared, heat, etc.) is carefully exploited and fused for video-surveillance applications. These sensors are usually used for 24-hour monitoring of indoor or outdoor environments, such as low luminosity and suburbs.

The types of agents also vary according to their application scenarios, mainly including but not limited to the following aspects: price, capacity, energy supply mode, computing, and communication capabilities.

In the context considered in this thesis, we mainly focus on distributed estimation in multi-agent networks. Agents in the network cooperatively estimate certain parameters(or states) of the surrounding environment based on their local measurements. Due to the limited information obtained by each agent (limitation in space or time, singleness of sensor types, etc.), every agent needs to cooperate with others to obtain reliable estimation results.

1.2 Filtering

Filtering problem consists of estimating the internal state of a dynamic system when partial measurements are made and random disturbances are present in the sensor and dynamic system. In a linear system with Gaussian noise, the Kalman filter(KF) [6] can be exploited to calculate the mean vector and covariance matrix of the estimated parameters recursively since the predicted posterior, as well as the posterior, are both Gaussian. For nonlinear and non-Gaussian cases, extended Kalman filter(EKF) and unscented Kalman filter(UKF) [6] are proposed as suboptimal solutions though the approximation procedures may lead to large errors and even divergence in final results. In the early 1990s, particle filtering turned out to be the preferred method for nonlinear and non-Gaussian systems [6]. It is a Monte Carlo approximation of optimal sequential Bayesian estimation. Therefore, in this thesis, we focus our research on particle filter(PF) due to the reason that the physical systems consist of massive nonlinear and non-Gaussian elements in many applications of multi-agent networks.

1.2.1 Bayesian Filtering

Dynamic state estimation refers to estimating the state of a time-varying system under a sequence of noisy measurements. In this thesis, we use the state-space approach to model dynamic systems. The state vector, which is the focus of the state-space approach to time-series modeling, contains all the relevant information about the system. For example, in the tracking problem, the state vector includes some necessary kinematic

characteristics of the target(e.g., velocities, accelerations). The measurement vector stacks the measurements that relate to the state.

In order to infer the system state from the measurements, two models are necessary: the transition model and the measurement model. The transition model describes the evolution process of the state with time while the measurement model explains the relationship between state vector and measurements. Bayesian filtering [7] is one of the most commonly used tools in state estimation.

From a Bayesian perspective, dynamic state estimation is the process of computing the confidence of the current state vector given a sequence of measurements. The confidence is also called the posterior probability density(pdf) function. Such a process contains two essential stages: prediction and update. The prediction stage utilizes the information of the transition model and the estimated state vector at the last time instant to predict the pdf of current state vector(see (1.3)). The update stage tries to update the pdf by incorporating the measurement information. This process is usually implemented based on Bayes' theorem(see (1.4)).

Assume we have a system and we aim to estimate its state vector over time. The transition model(1.1) and measurement model(1.2) are stated in below.

$$\boldsymbol{x}_n = \boldsymbol{g}_n(\boldsymbol{x}_n, \boldsymbol{u}_n) \tag{1.1}$$

$$\boldsymbol{y}_n = \boldsymbol{h}_n(\boldsymbol{x}_n, \boldsymbol{v}_n) \tag{1.2}$$

In (1.1), $\boldsymbol{x}_n$ is the state vector at time instant n, $\boldsymbol{g}_n$ is the possibly nonlinear transition function at time n, $\boldsymbol{u}_n$ is an i.i.d. process noise vector. In (1.2), $\boldsymbol{y}_n$ is the measurement vector at time n, $\boldsymbol{h}_n$ is the possibly nonlinear measurement function at time instant n, $\boldsymbol{v}_n$ is an i.i.d. measurement noise vector.

The goal of filtering is to estimate state vector $\boldsymbol{x}_n$ based on the set of measurements $\boldsymbol{y}_{1:n} = \{\boldsymbol{y}_i, i = 1, \cdots, n\}$ up to time n. In order to do this, we construct the posterior pdf $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$.

In the prediction stage, the posterior pdf at last time instant $p(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})$ is known. At the initial stage when $n = 0$, it is assumed that the initial pdf $p(\boldsymbol{x}_0|\boldsymbol{y}_0) = p(\boldsymbol{x}_0)$ and is known in advance. The prior pdf of the state can be computed via the following Chapman-Kolmogorov equation:

$$p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \int p(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})p(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})d\boldsymbol{x}_{n-1}. \tag{1.3}$$

After measurement $\boldsymbol{y}_n$ becomes available, we enter the update stage. The posterior pdf can be acquired via Bayes' theorem:

$$p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) = \frac{p(\boldsymbol{y}_n|\boldsymbol{x}_n)p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})}{p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})}, \tag{1.4}$$

where the normalization term in denominator ensures the integral of $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$ with respect to $\boldsymbol{x}_n$ equals 1 and can be calculated by

$$p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1}) = \int p(\boldsymbol{y}_n|\boldsymbol{x}_n)p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})d\boldsymbol{x}_n. \tag{1.5}$$

The statistics of both process noise and measurement noise are assumed known and the (1.3) and (1.4) form the basis of Bayesian filtering. However, this recursive propagation of the posterior density is only a conceptual solution, and in general we cannot give a closed-form solution when a nonlinear and non-Gaussian state-space model is adopted. Particle filtering is proposed to solve this problem.

1.2.2 Particle Filtering

Particle filtering is a method that uses a set of weighted particles to represent the posterior distribution of the state vector to be estimated. As the number of particles goes to infinity, the acquired description of the posterior approaches the optimal Bayesian estimate. The following derivation comes from reference [7].

For the convenience of explanation, we use $\{\boldsymbol{x}_{0:n}^m, \omega_n^m\}_{m=1}^M$ to describe the posterior pdf $p(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n})$ in a non-parametric way, where $\{\boldsymbol{x}_{0:n}^m, m = 1, \cdots, M\}$ is a set of particles with corresponding weights $\{\omega_n^m, m = 1, \cdots, M\}$. M is the number of particles. The weights should be normalized, which means $\sum_{m=1}^M \omega_n^m = 1$. To make it clear, $\boldsymbol{x}_{0:n}$ is the set of all state vectors up to time n. Then the posterior pdf can be approximated as

$$p(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n}) \approx \sum_{m=1}^{M} \omega_n^m \delta(\boldsymbol{x}_{0:n} - \boldsymbol{x}_{0:n}^m). \tag{1.6}$$

The weights of particles are calculated based on importance sampling(IS). Importance sampling addresses the inability to sample from the target distribution. Assuming the samples $\boldsymbol{x}_{0:n}^m$ are drawn from an importance density $q(\boldsymbol{x})$, which can be any form of distribution, the weight defined in (1.6) should looks like

$$\omega(\boldsymbol{x}_{0:n}^m) \propto \frac{p(\boldsymbol{x}_{0:n}^m|\boldsymbol{y}_{1:n})}{q(\boldsymbol{x}_{0:n}^m|\boldsymbol{y}_{1:n})}. \tag{1.7}$$

Our goal is to approximate $p(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n})$ given $p(\boldsymbol{x}_{0:n-1}|\boldsymbol{y}_{1:n-1})$ and a new set of particles at time n. If the importance density is factorized as

$$q(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n}) = q(\boldsymbol{x}_{0:n-1}|\boldsymbol{y}_{1:n-1}) q(\boldsymbol{x}_n|\boldsymbol{x}_{0:n-1}, \boldsymbol{y}_{1:n}), \tag{1.8}$$

we can obtain sample $\boldsymbol{x}_{0:n}^m$ from distribution $q(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n})$ by appending a new state vector $\boldsymbol{x}_n^m$ sampling from $q(\boldsymbol{x}_n|\boldsymbol{x}_{0:n-1}, \boldsymbol{y}_{1:n})$ to an already existing sequence state $\boldsymbol{x}_{0:n-1}^m$. The benefit of doing so is that the weights per time instant can be calculated recursively, and hence it is called sequential importance sampling(SIS).

$$\begin{aligned}
p(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n}) &= \frac{p(\boldsymbol{y}_n|\boldsymbol{x}_{0:n}, \boldsymbol{y}_{1:n-1}) p(\boldsymbol{x}_{0:n}|\boldsymbol{y}_{1:n-1})}{p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})} \\
&= \frac{p(\boldsymbol{y}_n|\boldsymbol{x}_{0:n}, \boldsymbol{y}_{1:n-1}) p(\boldsymbol{x}_n|\boldsymbol{x}_{0:n-1}, \boldsymbol{y}_{1:n-1}) p(\boldsymbol{x}_{0:n-1}|\boldsymbol{y}_{1:n-1})}{p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})} \\
&= \frac{p(\boldsymbol{y}_n|\boldsymbol{x}_n) p(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) p(\boldsymbol{x}_{0:n-1}|\boldsymbol{y}_{1:n-1})}{p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})} \\
&\propto p(\boldsymbol{y}_n|\boldsymbol{x}_n) p(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) p(\boldsymbol{x}_{0:n-1}|\boldsymbol{y}_{1:n-1})
\end{aligned} \tag{1.9}$$

By substituting (1.8) and (1.9) into (1.7), the recursive form of the calculation of particle weight can be rewritten as

$$\begin{aligned}\omega(\boldsymbol{x}_{0:n}^m) &\propto \frac{p(\boldsymbol{x}_{0:n}^m|\boldsymbol{y}_{1:n})}{q(\boldsymbol{x}_{0:n}^m|\boldsymbol{y}_{1:n})} \\ &\propto \frac{p(\boldsymbol{y}_n|\boldsymbol{x}_n^m)p(\boldsymbol{x}_n^m|\boldsymbol{x}_{n-1}^m)p(\boldsymbol{x}_{0:n-1}^m|\boldsymbol{y}_{1:n-1})}{q(\boldsymbol{x}_n^m|\boldsymbol{x}_{0:n-1}^m,\boldsymbol{y}_{1:n})q(\boldsymbol{x}_{0:n-1}^m|\boldsymbol{y}_{1:n-1})} \\ &\propto \omega(\boldsymbol{x}_{0:n-1}^m)\frac{p(\boldsymbol{y}_n|\boldsymbol{x}_n^m)p(\boldsymbol{x}_n^m|\boldsymbol{x}_{n-1}^m)}{q(\boldsymbol{x}_n^m|\boldsymbol{x}_{0:n-1}^m,\boldsymbol{y}_{1:n})}.\end{aligned} \tag{1.10}$$

Furthermore, if $q(\boldsymbol{x}_n|\boldsymbol{x}_{0:n-1},\boldsymbol{y}_{1:n}) = q(\boldsymbol{x}_n|\boldsymbol{x}_{n-1},\boldsymbol{y}_n)$ holds, then only $\boldsymbol{x}_n^m$ need be stored.

So far, the weight calculation formula can be simplified as

$$\omega(\boldsymbol{x}_n^m) \propto \omega(\boldsymbol{x}_{n-1}^m)\frac{p(\boldsymbol{y}_n|\boldsymbol{x}_n^m)p(\boldsymbol{x}_n^m|\boldsymbol{x}_{n-1}^m)}{q(\boldsymbol{x}_n^m|\boldsymbol{x}_{n-1}^m,\boldsymbol{y}_n)}, \tag{1.11}$$

and the posterior pdf $p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n})$ can be approximated using (1.6).

The pseudo-code of the SIS particle filtering is given by Algorithm 1. Besides, SIS particle filters have a common problem called particle impoverishment, which means that most particles will have negligible weights after several iterations, resulting in a large performance degradation. This can be alleviated by introducing resampling algorithms [8].

Algorithm 1 SIS Particle Filtering(PF)

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M$, $\boldsymbol{y}_n$
1: **for** $m = 1, \cdots, M$ **do**
2: Draw $\boldsymbol{x}_n^m \sim q(\boldsymbol{x}_{n-1}|\boldsymbol{x}_{n-1}^m, \boldsymbol{y}_n)$
3: Calculate the corresponding weight, ω_n^m, using (1.11)
4: **end for**
5: **return** $\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M$

1.3 Scenario

In this thesis, the target tracking problem is used in the simulation part to check the algorithms' performance, and in this Section, we are going to introduce the specific scenario we build, which is based on the Wiener process acceleration model [9].

We assume the state vector(1.12) of a certain target contains 6 elements, which are the position, velocity and acceleration in two-dimensional space.

$$\boldsymbol{x}_n = [x_{n,1} \quad x_{n,2} \quad \dot{x}_{n,1} \quad \dot{x}_{n,2} \quad \ddot{x}_{n,1} \quad \ddot{x}_{n,2}]^T \tag{1.12}$$

The target moves according to a specified transition function(1.13).

$$\boldsymbol{g}(\boldsymbol{x}_n) = \boldsymbol{D} \cdot \boldsymbol{x}_n + \boldsymbol{u}_n \tag{1.13}$$

The transition function is linear with respect to state vector and the matrix $\boldsymbol{D}$ is time independent, hence we omit the subscript n of function $\boldsymbol{g}$.

$$\boldsymbol{D} = \begin{bmatrix} 1 & 0 & t & 0 & \frac{1}{2}t^2 & 0 \\ 0 & 1 & 0 & t & 0 & \frac{1}{2}t^2 \\ 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

t is the state transition interval. $\boldsymbol{u}_n$ follows a multivariate Gaussian distribution $\mathcal{N}(0, \boldsymbol{R})$, where

$$\boldsymbol{R} = \sigma_u^2 \begin{bmatrix} \frac{1}{20}t^5 & 0 & \frac{1}{8}t^4 & 0 & \frac{1}{6}t^3 & 0 \\ 0 & \frac{1}{20}t^5 & 0 & \frac{1}{8}t^4 & 0 & \frac{1}{6}t^3 \\ \frac{1}{8}t^4 & 0 & \frac{1}{3}t^3 & 0 & \frac{1}{2}t^2 & 0 \\ 0 & \frac{1}{8}t^4 & 0 & \frac{1}{3}t^3 & 0 & \frac{1}{2}t^2 \\ \frac{1}{6}t^3 & 0 & \frac{1}{2}t^2 & 0 & t & 0 \\ 0 & \frac{1}{6}t^3 & 0 & \frac{1}{2}t^2 & 0 & t \end{bmatrix}. \tag{1.15}$$

To estimate the unknown time-varying state vector of the moving target, we deploy 9 agents(whose positions are known) in two-dimensional space to perform some measurements, including two quantities(known): the target's range and Doppler (range rate). We assume that the position of the k^{th} sensor is described by $\boldsymbol{l}_k = (l_{k,1}, l_{k,2})$, then the measurement vector can be expressed as

$$\boldsymbol{h}_k(\boldsymbol{x}_n) = [h_{k,range}(\boldsymbol{x}_n) \quad h_{k,doppler}(\boldsymbol{x}_n)]^T, \tag{1.16}$$

where the two elements are stated as follows,

$$h_{k,range}(\boldsymbol{x}_n) = \sqrt{(x_{n,1} - l_{k,1})^2 + (x_{n,2} - l_{k,2})^2}, \tag{1.17}$$

$$h_{k,doppler}(\boldsymbol{x}_n) = \frac{\dot{x}_{n,1}(x_{n,1} - l_{k,1}) + \dot{x}_{n,2}(x_{n,2} - l_{k,2})}{\sqrt{(x_{n,1} - l_{k,1})^2 + (x_{n,2} - l_{k,2})^2}}, \tag{1.18}$$

and the measurement noise $\boldsymbol{v}_{n,k}$ follows $\mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_w^2 \end{bmatrix})$.

The parameter settings of the scenario is given in Tab.1.1 and Fig.1.1 shows the sensor network and a realization of the target trajectory.

Table 1.1: Scenario setup

parameter	value
σ_u	0.5
σ_v	1
σ_w	1
t	1
time length	30
number of sensors	9
area size	200*200
origin state vector	$[0, 0, 4, 13, -1, -3]^T$

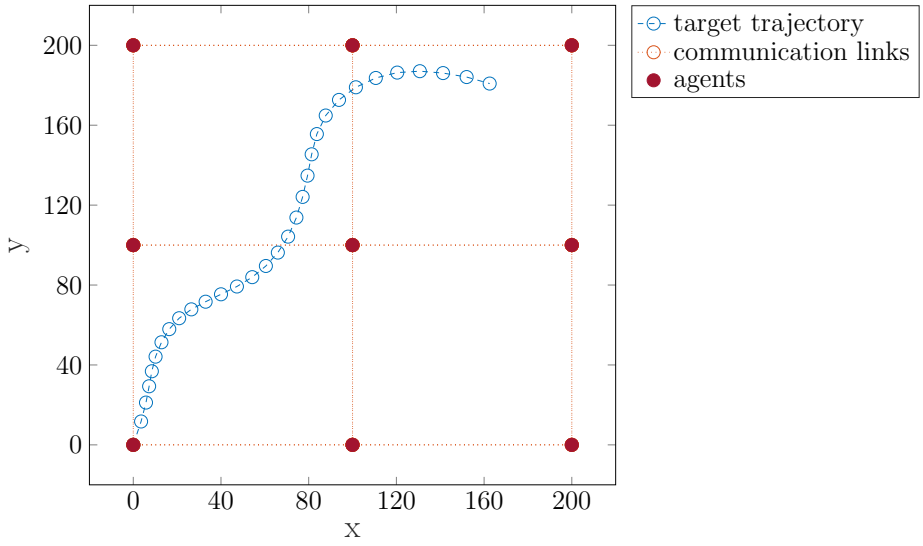


Figure 1.1: Multi-agent network, communication links and target trajectory(We here have 9 agents with known positions, and our goal is estimate the 6D state vector of the unknown agent. The dotted blue line is one unknown trajectory realization.)

1.4 Goals

The main goal of the thesis is to design energy-efficient distributed particle filtering algorithms(DPFs) for distributed estimation problems that occur in multi-agent networks. Energy consumption can be reduced from three directions: computation, communication, and memory. Our aim is to explore the possibility of directly exchanging particles between connected agents in a distributed network, as doing so minimizes the reliance on computational resources and strong assumptions. However, with the increase of the dimension of the state space, the direct exchange of particles will undoubtedly bring

great communication requirements. Therefore, how to select the exchanged particles and how to optimize the particle set have become the core of the research. The most canonical problem, target tracking, is used to test the performance of the proposed algorithms.

1.5 Outline

The rest of the thesis is organized as follows.

- In Chapter 2, a literature review of state-of-the-art DPFs will be presented first, followed by the reproduction work of two up-to-date DPFs. The above two algorithms are evaluated under the same simulation setup. A discussion of their performance comparison as well as limitations are also conducted.
- In Chapter 3, inspired by the work in [10], a Gaussian process enhanced DPF algorithm based on direct particle exchange is proposed.
- In Chapter 4, various metaheuristic optimization algorithms are used to find the global optimal particle set, so as to complete the fusion of the whole network information and improve the estimation performance.
- In Chapter 5, a comprehensive comparison of all algorithms presented in this thesis is given.
- In Chapter 6, the work in the thesis is concluded. The contributions as well as the future directions are discussed.

2 Distributed Particle Filtering

In this Chapter, we first present a literature review on DPF and then focus on reproducing two of the most up to date DPFs, comparing their tracking performance under the same simulation setup, and finally making a discussion on their contributions as well as limitations.

In Section 2.1, the literature review is given and in Section 2.2, we formulate the centralized particle filtering problem and extend it to the distributed manner. The first DPF, graph Laplacian distribute particle filtering(GL-DPF), proposed in [11] is introduced in Section 2.3 while the second DPF, Gaussian mixture distributed particle filtering(GM-DPF), proposed in [12] is introduced in Section 2.4. In Section 2.5, both of the above-mentioned DPFs are tested under the Wiener process acceleration model and the simulation results are presented. In Section 2.6, a comparison and discussion work has been conducted.

2.1 Literature Review

DPF arises due to the growing problems of distributed filtering occurring in massive nonlinear, non-Gaussian systems and the goal is to compute the global posterior probability. In [6], you can find a survey of the previous contribution to DPF in multi-agent networks by 2012. For distributed estimation algorithms, communication aspects of the underlying network always come first. To be more concrete, these aspects usually include the communication topology(how different agents are linked) and the properties of the communication links(e.g., latency, transmission rate, packet loss ratio). To effectively implement DPF, some approximation strategies must be utilized to reduce the burden on communication and computation to adapt to practical applications. Though there is no doubt that the estimation results would suffer from the degradation with the introduction of various approximation techniques, designing an effective approximation strategy for DPF to seek a balance between estimation performance and cost of energy(e.g., computation, communication) is of great importance.

Focusing on DPF, existing algorithms usually differ in the following aspects: the type of communication signals(e.g., likelihood values, posterior values), the amount of communication signals(in which many signals compressing techniques can be exploited) and so on. Based on different underlying communication topology, DPF can be classified into three categories, FC(Fusion center)-based DPF, LA(Leading agent)-based DPF, and consensus-based DPF(see Fig 2.1).

- FC-based DPF

  FC-based DPF requires a central processing unit and thus can not work in a fully distributed manner. In FC-based DPF, each agent would perform local particle

filtering and transmit the assigned information to the FC. The main disadvantage of FC-based DPF is its weak scalability and flexibility since every agent must be able to access the FC. If there is no direct link between an agent and the FC, multi-hop communication is then required, creating the need for an efficient routing protocol which also needs to be updated with time if the network topology is changing over time. To conclude, FC-based DPF is delicate to the change of network topology and is easy to break down when the fusion center does not work properly.

- LA-based DPF

  In LA-based DPF, only the currently active agents(the LAs) perform particle filtering and transmit the most up-to-date estimation information to the next new LAs, thus the information accumulates sequentially. Based on the number of LAs during the time interval, two different LA-based DPFs are investigated. In the so-called single LA case, there is only one active LA in every discrete time step. The single LA would perform particle filtering using its measurements, or possibly, the measurements from its neighbors, and then execute the communication step, transmitting the estimation to the selected new LA. In the aggregation chain(AC) case, multiple LAs are activated sequentially and form an AC. At a discrete-time instant n, LA selected first in the AC performs particle filtering only based on its measurements and transmits the results to the next selected LA. The latter one would come up with the updated estimation incorporating its measurement and the posterior information sent by the former agent. In the end, the last LA in the AC can hereby generate a solution reflecting the measurements of all the previous LA in the nth AC as well as all previous ACs.

- Consensus-based DPF

  In the consensus-based DPF, all agents perform particle filtering simultaneously and would possess a particle representation of the global estimation after the communication stage. The common goal is achieved by exploiting various consensus algorithms to help agree on certain quantities across the network. Consensus-based DPF is attractive in recent years due to the following reasons. First, it only requires local communications between connected agents which makes it more robust to the change of the network topology and eliminates the need for routing protocol design. Second, each agent would finally possesses a global estimate, and it may be important for those applications where agents need to perform some actions based on the estimation results. Third, it is also robust for link failure and single-agent damage. However, the biggest challenge is its large communication cost. Overlarge consensus iterations will also make it not suitable for real-time applications.

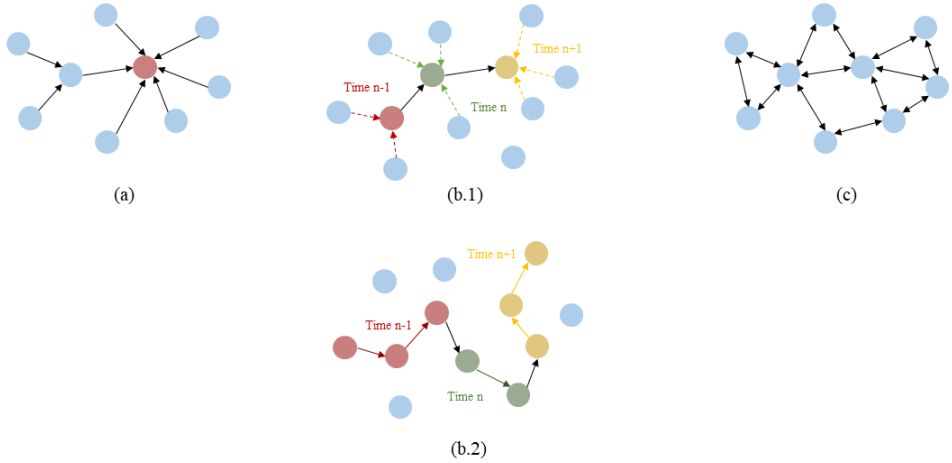


Figure 2.1: (a) FC-based DPF(the red node represents the FC and black arrows represent communication links), (b.1) LA-based DPF: Single LA case(black arrows represent transmissions between consecutive LA, dashed colored arrows represent transmissions from neighboring nodes to the certain LA, and different colors indicate different time steps), (b.2) LA-based DPF: AC case(black arrows represent transmissions from the last LA in previous AC to the first LA in current AC, colored arrows represents transmissions between consecutive LAs inside the same AC, and different colors indicate different time steps and different ACs as well), (c) Consensus-based DPF(all agents communicate with their neighboring agents simultaneously and thus the black arrows are bidirectional).

This thesis focuses on the full distribution scheme of particle filtering, hence the FC-based DPF and LA-based DPF are excluded. Considering the parameters communicated in the consensus algorithm, we could divide consensus-based DPFs into the following three types according to [6]. (i) consensus-based calculation of particle weights; (ii) consensus-based calculation of posterior parameters; (iii) consensus-based calculation of likelihood parameters. In the next section, the above three kinds of consensus-based DPF are further explained. Table 2.1 summarizes the existing up to date researches on fully distributed DPFs.

2.1.1 Consensus-based DPF

- Consensus-based Calculation of Particle Weights

  This method is based on factorizing the global likelihood function into multiple local likelihood functions. Based on common statistical assumptions, measurements taken at different sensors are conditionally independent given the state vector. The detailed derivation can be found in [6]. The advantage of this al-

gorithm lies in its simple theoretical derivation and proof. However, in practical implementation, it has several limitations.

  - Since the core idea is to exchange the particle weights, the prerequisite is to ensure the identical set of particles in every agent and it, in turn, requires the local random number generators at agents to be synchronized. However, it is often not an easy task to synchronize all the agents in a large WSN.
  - The communication overhead is proportional to the amount of particles that dominates the estimate accuracy. To save communication overhead while maintaining acceptable performance, lots of signal approximation algorithms are proposed. In [11], a Graph Laplacian Signal Approximation algorithm is proposed to approximate the particle weights by exploiting the inner structure of particle distribution. In [13], an auxiliary particle filter is used to replace the bootstrap particle filer, making it possible to require fewer particles with the help of the more delicate design of the importance sampling function, and selective gossip is used to further reduce communication cost. A robust scheme for DPF is presented in [14] where an "M-posterior" DPF method based on the Weiszfeld algorithm is proposed to reduce the size of the particle set. In [14], an additional communication step is again adopted which comprises an extended communication range when an estimator is required at a specific sensor. Algorithms [15] [16] which are recently developed to compress the particle set while guaranteeing the filtering performance can also be applied to the distributed scheme to reduce the communication overhead.

- Consensus-based Calculation of Likelihood Parameters

  The goal of the filter is to obtain the posterior probability distribution of the state vector given measurements. The global posterior probability can be calculated at each node when giving the form of the global likelihood function(GLF) [17]. The idea behind this kind of method is to compute or approximate the global likelihood function with the help of a consensus algorithm. Since every agent can acquire the GLP after some computation work and consensus algorithm, one of the benefits is that the local agent can compute the estimated states independently without the acquirement of synchronization when drawing particles. Another great advantage is that the quantities in consensus algorithms are parameters describing the local likelihood functions, thus uncorrelated to the particle numbers, bringing a great reduction in communication cost. In [17], the algorithms only work under the prerequisite that the local likelihood functions belong to the exponential family. In [18], authors extend the exponential family local likelihood function to a more general case.

- Consensus-based Calculation of Posterior Parameters

  The idea of particle filtering is using the Monte-Carlo method to approximate the posterior probability in a non-parametric way by introducing a great number of particles. Here, we use the parametric approximation to describe the local posterior and exchange these parameters instead of thousands of particles to reduce

communication overhead but sacrifice some accuracy as a cost. In [19] [20], posteriors are approximated with Gaussian distributions which can be described with only 2 variables, the mean and covariance matrix respectively. Since Gaussian approximation is always too strong and not suitable for the nonlinear and non-Gaussian applications, the Gaussian mixture model(GMM) is hereby introduced to fit any kind of posterior distributions with an adjustable number of Gaussian components. In [21], the GMM is used to represent the posterior function and the distributed Expectation Maximization(EM) algorithm is adopted to fuse the local information. However, the algorithm limits the number of mixed Gaussian to be the same at each agent, and the fusion process is constrained to be linear. An adaptive Gaussian mixture learning algorithm for DPF is proposed in [22], aiming at adaptively choosing the Gaussian components in GMM for each local agent. And in [12], an importance sampling based nonlinear fusion technique from the optimal perspective is proposed.

2.1.2 Consensus vs. Diffusion

Apart from the consensus algorithm, diffusion strategy is also adopted in DPF in recent years. The major problem of consensus-based strategy is its two time-scale implementations: one for measurements and the other for communications. Usually, the agents need to go through a sufficient number of iterations in the second time-scale to make the network reach the consensus which is impractical in online filtering. Therefore, diffusion strategy is taken into account due to their nature of operating under a single time-scale and the consensus doesn't have to be reached during two consecutive measurements. By local communication among adjacent agents, the information would gradually diffuses through the whole network. Generally, the diffusion strategy consists of two phases at each time instant [23]: (a) during the adaptation phase, each agent updates its local estimates with the measurements of its neighbors; (b) during the combination phase, the neighbors' estimates are merged. A Bayesian interpretation of distributed diffusion filtering algorithms can be found in [24]. In [25] [26], the adapt-then-combine(ATC) diffusion strategy is introduced to distributed particle filtering.

2.1.3 Synchronous vs. Asynchronous

In distributed networks, synchronization is always a big issue. All the papers presented above assume the agent network operates synchronously. It means measurements at all sensors are performed simultaneously and the next stage begins after all agents have finished their computation and communication tasks. These two assumptions are often too strict in large networks. For the asynchronous measurements, there are two main approaches. In [27], the extrapolation strategy(ES) is proposed. As its name suggests, an extrapolation procedure is performed to fuse the measurements from neighbors and a delay estimator is developed to assist the extrapolation. The disadvantage of this method is it is too computational intense especially when the particle size is relatively large. In [28], another algorithm called asynchronous batch estimation (ABE) is used to tackle the asynchronous issue and saves a lot of computation energy. For the second synchronization problem, there are two options. The first one is the whole network

enters the next time instant until the slowest agent has completed all its tasks. By doing this, the algorithm would be simpler to implement and analyze. But on the contrary, a lot of time will be wasted if the computing power of different agents varies greatly. The other option is when an agent completes its current task, it goes directly to the next stage without waiting for the processing progress of other agents in the network. Adopting this strategy usually results in faster convergence but the reliability still needs to be further considered.

Table 2.1: Distributed Particle Filtering - Algorithm

Authors	Quantities	Strategies	Synchronous	Remark
Gu et al. [21]	posterior	consensus	✓	GMM approximation
Mohammadi et al. [20]	posterior	consensus	✓	Gaussian approximation
Li et al. [12]	posterior	consensus	✓	GMM approximation
Mohammadi et al. [25]	posterior	diffusion	✓	Gaussian approximation
Vázquez et al. [14]	weights	diffusion	✓	"M-posterior" algorithm
Song et al. [26]	posterior	diffusion	✓	Dynamic Event-triggered
Rabbat et al. [11]	weights	consensus	✓	Graph Laplacian
Üstebay et al. [13]	weights	consensus	✓	selective gossip
Hlinka et al. [17]	likelihood	consensus	✓	only exponential family
Hlinka et al. [18]	likelihood	consensus	✓	extends to general functions
Mohammadi et al. [29]	likelihood	consensus	✓	consensus+innovations
Hlinka et al. [27]	posterior	consensus	×	ES
Li et al. [28]	likelihood	consensus	×	ABE

2.2 From PF To DPF

Different with PF, DPF aims at approaching the global posterior distribution $p(\boldsymbol{x}_n|\boldsymbol{y}_n)$ at every time instant n for each agent k. $\boldsymbol{y}_n$ is the global measurement vector and it is now given by the collection of all local measurement vectors, i.e., $\boldsymbol{y}_n = (\boldsymbol{y}_{n,1}^T, \cdots, \boldsymbol{y}_{n,K}^T)^T$.

In one of the most popular particle filters, Sampling Importance Resampling Filter (SIR) [30], the importance density $q(\cdot)$ is chosen to be the state transition probability density function. The update equation of $\omega(1.11)$ then becomes

$$\omega(\boldsymbol{x}_{0:n}^m) \propto p(\boldsymbol{y}_n|\boldsymbol{x}_n^m) \times \omega(\boldsymbol{x}_{0:n-1}^m). \tag{2.1}$$

If resampling is performed in every single step, since each particle holds the same weight after the resampling step, we can even omit the weight value of the particle at the previous moment when we compute the weight value at the current time instant.

In addition, for distributed sequential Bayesian estimation, we often assume measurement noise at different sensors is uncorrelated and the following assumption holds.

Assumption : [6] *The measurements taken at different sensors at each time instant n are conditionally independent given the state vector $\boldsymbol{x_n}$. And the global likelihood function can be factorized into a product of local likelihood functions*(2.2):

$$p(\boldsymbol{y}_n|\boldsymbol{x}_n) = \prod_{k=1}^{K} p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n), \tag{2.2}$$

where $\boldsymbol{y}_{n,k}$ denotes measurement taken at nth time instant by agent k.

We have already introduced some state-of-the-art DPFs in the previous Section. First, consider the case in which particle weights are transmitted through the sensor network to get the global estimation. Since the performance of the particle filter is closely related to the number of particles, to obtain satisfactory results, drastically increasing the number of particles often results in unbearable communication overhead. In [11], the graph Laplacian approximation technique is exploited to reduce the size of transmitted data (the transmitted data would be particles' weights if no compression is made, and the size would equal to the number of particles) while keeping the estimation performance at a good level. To further save the communication bandwidth, in [12], the author approximates the posteriors as a Gaussian mixture and transmits sufficient statistics instead of particle weights to cut the cost considerably. Since the above two DPFs fall into different categories and both show good performance, in the following, we would first explain the algorithms in detail and then make a comprehensive comparison from the communication and computation aspects.

2.3 Graph Laplacian Distributed Particle Filtering

In this Section, we would briefly introduce the core idea of the graph Laplacian distributed particle filtering(GL-DPF) proposed in [11]. Since it belongs to the category of consensus-based calculation of particle weights, to fuse the information, we need to make sure each sensor generates the same particle set at every time instant. Besides, we also apply the *log* operation to (2.2) to transfer the geometric averaging to the arithmetic averaging to enable a consensus algorithm.

$$p(\boldsymbol{y}_n|\boldsymbol{x}_n) = \exp\{\sum_{k=1}^{K} \log(p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n))\} \tag{2.3}$$

Based on the global likelihood function (2.3), we can hereby calculate the global particles' weights incorporating all information from the sensor network.

Various kinds of distributed consensus algorithms then can be exploited to compute the term $p(\boldsymbol{y}_n|\boldsymbol{x}_n)$ by exchanging information with neighbors. We used the randomized gossip algorithm [31] in our simulation. One more thing that needs to be mentioned is that the total number of sensors has to be known in advance.

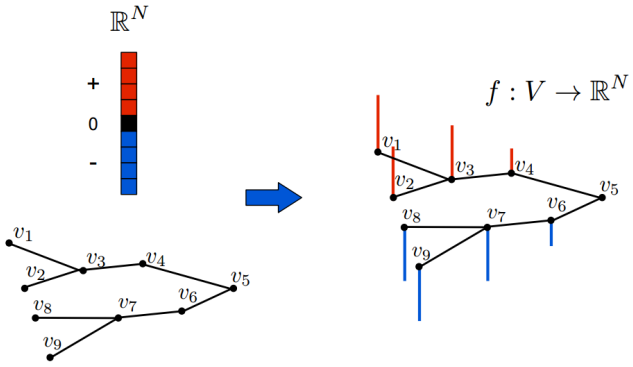


Figure 2.2: A simple example of graph signal.

2.3.1 Graph Laplacian Approximation

We first briefly introduce the Graph signal which provides a nice compact format to encode structure within the data. And with some operations, it is possible to compress the signal by exploiting its inherent structure. The tutorial can be found in [32].

Fig 2.2 gives an intuitive illustration of graph signal. The edges together form the structure while the values at vertices represent the data. The core idea of GL-DPF is to compress the weights of M particles into $z(z < M)$ scalars so that z scalars are transmitted in the communication process instead of transmitting particle weights. We first build a graph by taking particles as vertices and the corresponding weights as signals attached to each vertex. It is reasonable to think that neighboring particles should have similar weights and the graph signal is a good way to take this structure within data into account. Then we place edges between particles under the guidance of the k-nearest neighbors algorithm(k-NN) [33], which means each particle would connect to its nearest k particles. So far, we have already successfully constructed the underlying graph structure. The Laplacian matrix $\mathbf{L} \in \mathbb{R}^{M \times M}$ of the graph, defined by $\mathbf{D}$(Degree Matrix)-$\mathbf{A}$(Adjacent matrix), is the approximation of the Laplace operator and provides a notion of 'frequency' on graphs. If we multiply the $\mathbf{L}$ matrix with the data vector $\boldsymbol{f}$(which will be the particles' weights in our case), we get

$$\boldsymbol{f}^T \mathbf{L} \boldsymbol{f} = \frac{1}{2} \sum_{i,j=1}^{M} L_{ij} (\boldsymbol{f}(i) - \boldsymbol{f}(j))^2, \tag{2.4}$$

where L_{ij} is the entry of the i^{th} row, j^{th} column in matrix $\mathbf{L}$ and M is the number of particles. Intuitively, (2.4) gives a measure of "smoothness". To analyze the signal from the frequency(smoothness) perception, we first need to do an eigenvalue decomposition(EVD) on graph Laplacian matrix. Since $\mathbf{L}$ is a symmetric matrix, the process can be denoted as $\mathbf{L} = \mathcal{X} \mathbf{\Lambda} \mathcal{X}^T$, where columns in matrix $\mathcal{X}$ are right eigenvectors, and the diagonal elements in $\mathbf{\Lambda}$ are the corresponding eigenvalues. Eigenvectors with smaller eigenvalues vary less rapidly through the graph. Based on the above theory analysis, we are now able to import the concept of Graph Fourier Transform(GFT).

GFT:

$$\hat{\boldsymbol{f}} = \mathcal{X}^T \boldsymbol{f} \tag{2.5}$$

IGFT(Inverse GFT):

$$\boldsymbol{f} = \mathcal{X} \hat{\boldsymbol{f}} \tag{2.6}$$

In practical applications, instead of transmitting the entire signal, we could approximate the signal with frequency components to save communication overhead. However, the prerequisite is that each agent should know the eigenvector matrix $\mathcal{X}$ to recover the signal from the frequency domain, putting constraint on the identical particle set for all agents. In [11], the author reserved the low-frequency part of the original signal by taking z smallest Laplacian eigenvalues(Laplacian transform coefficients).

2.3.2 Algorithm

In this Section, the pseudo-code of GL-DPF algorithm is presented in Algorithm 2.

At the end of time instant $n-1$, we assume every agent holds an identical particle set, denoted as $\{\boldsymbol{x}_{n-1}^m\}_{m=1}^M$. After the measurements at time instant n become available, we need to apply the GL-DPF to obtain an updated particle set $\{\boldsymbol{x}_n^m\}_{m=1}^M$ integrating the information of the whole sensor network for each agent. First, every agent performs the local particle filter and end up with a new weighted particle set $\{\boldsymbol{x}_n^m, \omega_{n,k}^m\}_{m=1}^M$, where k means the k^{th} agent. Second, graph Laplacian approximation is exploited to compress the weight information, including building the graph signal(with the help of k-NN algorithm) as well as the GFT operation. Then the weights of M particles are compressed into $z(z < M)$ Laplacian transform coefficients for each agent(corresponds to each column in $\hat{\boldsymbol{F}}_n$). Third, we use the randomized gossip algorithm [31] to compute the formula (2.3), which is achieved by randomly selecting two connected agents and averaging their Laplacian transform coefficients in each iteration. Since the convergence of the average gossip is asymptotic, and we need all agents to have exactly the same estimate, we then apply the max-consensus algorithm, which can be shown to reach the same maximum value for all agents after a finite number of iterations [34]. After the consensus is reached, every agent needs to perform the IGFT to recover the particle weight information from the Laplacian transform coefficients which is denoted by $\hat{\boldsymbol{f}}_n$ in the pseudo-code. Finally, the resampling algorithm is performed to obtain a particle set $\{\boldsymbol{x}_n^m\}_{m=1}^M$ with a weight equal to $1/M$. Note, the standard resampling algorithm through this thesis(function **Resample** in pseudo-code) refers to systematic resampling in [7]. The basic idea of systematic resampling is to concentrate on particles with large weights while abandon particles with small weights to prevent particle impoverishment. The implementation of this algorithm is shown in Algorithm 3.

Algorithm 2 GL-DPF

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M$, $\{\boldsymbol{y}_{n,k}\}_{k=1}^K$, execute at all agents $k = 1, \cdots, K$ in parallel:

Initialize M, z, L_g, N_{knn}(the number of nearest neighbors in k-NN)
$\{\boldsymbol{x}_n^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$
Calculate the weight matrix $\boldsymbol{W}_n$, where entry $\boldsymbol{W}_n^{mk} = \log(\omega_{n,k}^m)$
Build graph for particles(based on k-NN)
Calculate the Laplacian matrix $\boldsymbol{L}$ and perform the EVD: $\boldsymbol{L} = \mathcal{X}\Lambda\mathcal{X}^T$
Perform GFT: $\boldsymbol{F}_n = \mathcal{X}^T\boldsymbol{W}_n$
Approximate $\boldsymbol{F}_n$ to $\hat{\boldsymbol{F}}_n$(extract z eigenvalues)
$\hat{\boldsymbol{f}}_n = \text{GOSSIP}(\hat{\boldsymbol{F}}_n, L_g)$
Recover $\hat{\boldsymbol{W}}_n = \mathcal{X}\hat{\boldsymbol{f}}_n$
for $m = 1, \cdots, M$ **do**
    $\omega_n^m = \frac{\exp(\hat{\boldsymbol{f}}_n^m)}{\sum_{j=1}^M \exp(\hat{\boldsymbol{f}}_n^j)}$
end for
$\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M = \mathbf{Resample}(\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M)$
return $\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M$

Algorithm 3 Resample(Systematic Resampling)

Require: $\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M$

Initialize $M, c_1 = 0$
for $m = 2, \cdots, M$ **do**
    Construct CDF: $c_m = c_{m-1} + \omega_n^m$
end for
Start at the bottom of the CDF: $m = 1$
Draw a starting point: $u_1 \sim \mathcal{U}_{[0, M^{-1}]}$
for $\hat{m} = 1, \cdots, M$ **do**
    Move along the CDF: $u_{\hat{m}} = u_1 + M^{-1}(\hat{m} - 1)$
    while $u_{\hat{m}} > c_m$ **do**
        $m = m + 1$
    end while
    $\boldsymbol{x}_n^{\hat{m}} = \boldsymbol{x}_n^m$
    $\omega_n^{\hat{m}} = M^{-1}$
end for
return $\{\boldsymbol{x}_n^{\hat{m}}, \omega_n^{\hat{m}}\}_{m=1}^M$

2.4 Gaussian Mixture Distributed Particle Filtering

In this Section, we would briefly introduce the core idea of the Gaussian mixture distributed particle filtering(GM-DPF) proposed in [12]. Different with GL-DPF, it belongs to the category of consensus-based calculation of posterior parameter. To facilitate the introduction of GM-DPF later, we roughly divide it into two sub-steps, consensus and recovery respectively. The following derivation follows (2.2).

Considering the conditional independence of a Markov chain, the local likelihood

can be written as

$$p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n) = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n, \boldsymbol{y}_{1:n-1}) \tag{2.7}$$

Using the Bayes's theorem, it can be further factorized as

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

Finally we get the global posterior, written as a function of local posterior,

$$p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) = \frac{\prod_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})p(\boldsymbol{y}_{n,k}|\boldsymbol{y}_{1:n-1})}{p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}p(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1})} \tag{2.9}$$

$$p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \propto \frac{\prod_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})}{p(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}} \tag{2.10}$$

In the following, we use η_k to represent $p(\boldsymbol{x}_n|\boldsymbol{y}_{n,k}, \boldsymbol{y}_{1:n-1})$.

The remaining problem is to calculate the geometric averaging of local posterior distributions as well as the denominator. To avoid transmitting particles, Li et al. [12] use Gaussian Mixture Model to parametrically describe the local posterior and instead exchange the distribution statistics.

2.4.1 Gaussian Mixture Model

A Gaussian Mixture Model [35] is the convex combination of several Gaussian components, as shown in (2.11), where C is the number of Gaussian components, $\{\alpha_c, c = 1, \cdots, C\}$ are the mixture weights, and $\{\mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c),\ c = 1, \cdots, C\}$ are the Gaussian distributions. Each component is a multivariate Gaussian function. All the local, global as well as intermediary posteriors appearing in the algorithm would be approximated as Gaussian mixtures. Theoretically, the Gaussian mixture model can represent any kind of distribution.

$$GMM(\boldsymbol{x}_n) \approx \sum_{c=1}^{C} \alpha_c \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \tag{2.11}$$

The tracking performance of GM-DPF is closely related to the number of components in Gaussian mixture models. In this Section, we would make a summary of the techniques which are used to determine the number of Gaussian components and try some of them. Here we mainly focus on the finite mixture models, whereas for infinite mixture models, reference can be found in [36]. The expectation-maximization (EM) algorithm is a commonly used algorithm to fit the Gaussian mixture models. The advantages of the EM algorithm are listed as follows. First, it is the fastest algorithm to learn mixture models. Second, it can converge to the local optimal point. However, the drawbacks are also obvious. First, the EM algorithm is sensitive to the choice of initial points, which means the clustering result is not consistent. Second, we must set the number of Gaussian components in advance with the help of other external techniques.

Third, the singularities may occur if there are no sufficient points for every mixture, and one may have to regularize the covariance matrix artificially.

To help us determine the optimal number of Gaussian components, some information criteria in model selection [37] are proposed. We would introduce two of them, AIC and BIC respectively. AIC(i.e., Akaike's Information Criterion) selects the model with minimal

$$-2\log L(\Phi) + 2n_p \tag{2.12}$$

where Φ consists of the unknown parameters such as the component mean and covariance matrix, and the $\log L(\Phi)$ is the log-likelihood function of Φ. n_p equals the number of the total parameters we used in the mixture model. However, some researchers have found that AIC tends to overestimate the number of correct components.

The Bayesian Information Criterion(BIC) is presented in (2.13),

$$-2\log L(\Phi) + n_p \log n_m \tag{2.13}$$

where the additional variable n_m means the total number of measurements. Paper [38] also shows that under the normal mixture model case, the result of using BIC to select the appropriate number of components is consistent.

Although with the help of some information criteria, one can get a quite satisfying guess on the number of components, it is heavily computational since one must run the EM algorithm for every single guess. Besides, using information criteria doesn't avoid the singularity problem as well. To tackle the above problems, the variational Bayesian Gaussian mixture appeared. The variational Bayesian approach bypasses the singularity problem by enforcing prior probabilities on the involved parameters and resorting to the Bayes' theorem to perform the estimation. In [39], a way of applying the variational Bayesian approach to Gaussian mixture model approximation is presented. The author introduced an auxiliary latent random vector indicating from which component is every sample drawn. Besides, Gaussian and Wishart pdf are adopted as priors for mean values and covariance matrices respectively. According to the paper, it shows the number of mixtures can be determined by only giving a large enough number of components prior, meaning the algorithm can automatically achieve a tradeoff between model complexity and fitting performance.

2.4.2 Algorithm

We divide the process of computing the global posterior distribution into two sub-steps. First, the geometric mean of the local posteriors is computed, which we name the consensus step. Second, the global posterior is recovered by performing computations according to (2.10) using the prediction term and the result obtained in the first step. The second step is called the recovery step.

A. Consensus

Before the consensus step, each sensor should perform the local particle filtering and approximate the posterior as a Gaussian mixture. The Gaussian mixture learning algorithm based on EM is presented in Algorithm 4. Hence, the data transmitted by

sensors in the communication process is restricted to the Gaussian mixture statistics. In this step, the average consensus algorithm is used to compute the geometric averaging of local posteriors. We use η_k^i to represent the fusing posterior at sensor k at ith consensus iteration. As i approaches ∞, η_k^i approaches the result in centralized case. In every iteration, sensor k communicates with its adjacent sensors and calculates the fusing result according to (2.14)

$$\eta_k^{i+1}(\boldsymbol{x}_n) = \prod_{j \in \mathcal{N}_k} (\eta_j^i(\boldsymbol{x}_n))^{\epsilon_{k,j}}, \tag{2.14}$$

where $\eta_k^{i+1}(\boldsymbol{x}_n)$ is the posterior of state vector $\boldsymbol{x}_n$ at sensor k in the $(i+1)^{th}$ iteration of the average consensus algorithm at n^{th} time instant. $\mathcal{N}_k$ is the set of the neighboring sensors of sensor k, and $\epsilon_{k,j}$ is the Metropolis weight.

Definition 2.1: [40] *Metropolis weight*

$$\epsilon_{k,j} = \begin{cases} 1/\max(|\mathcal{N}_k|, |\mathcal{N}_j|), & if\ (k,j) \in \boldsymbol{E} \\ 1 - \sum_{l \in \mathcal{N}_k} \epsilon_{k,l}, & if\ k = j \\ 0, & otherwise \end{cases} \tag{2.15}$$

where $\boldsymbol{E}$ *means the collection of communication links.*

Since (2.14) involves the product of fractional powers of Gaussian mixtures, we are not able to calculate analytically. Therefore, importance sampling is adopted. At i^{th} iteration, sensor k obtains the Gaussian mixture statistics of its neighboring sensors after the communication step. The sensor first samples particles from each Gaussian mixture and then calculates the corresponding weights based on (2.16). For each Gaussian mixture at sensor $j \in \mathcal{N}_k$, the sampling particle size is proportional to the Metropolis weight. For example, the total sampling particles is restricted to M, then we draw $M_j = \lfloor M\epsilon_{k,j} \rfloor$ particles from $\eta_j^i(\boldsymbol{x}_n)$, where $\lfloor \cdot \rfloor$ is the floor function.

$$\omega_n^{j,m} = (\eta_j^i(\boldsymbol{x}_n^{j,m}))^{-1} \prod_{l \in \mathcal{N}_k} (\eta_l^i(\boldsymbol{x}_n^{j,m}))^{\epsilon_{k,l}} \tag{2.16}$$

After the final average consensus iteration, each sensor holds several sets of particle and weight pairs, and the Gaussian mixture learning algorithm(GML) is performed to fit the distribution. Furthermore, to save the consensus iterations and promote the convergence performance of the average consensus, an additional fine-tuning step is incorporated. In the fine-tuning process, similar components between different Gaussian mixtures are determined based on Kullback-Leibler divergence(KLD) criterion and then averaged in the parameter-based level.

Definition 2.2: [41] KLD *is a measure describing how similar a probability distribution* Q *is to a second reference probability distribution* P, *and it is calculated based on* (2.17).

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log(\frac{p(x)}{q(x)}) dx \tag{2.17}$$

In the fine-tuning process, the matched Gaussian components are used to form a new Gaussian distribution with the help of Gaussian approximation technique. The resulting Gaussian distribution is then scaled and serves as a new Gaussian component in the Gaussian mixture after fine-tuning. After the fine-tuning step, all sensors are expected to share the almost exact Gaussian mixture statistics. The pseudo-code for fine-tuning is presented in Algorithm 5.

B. Recovery

Now we assume that consensus is reached after infinity average iterations, then (2.10) can be reformulated as (2.18) and $\eta(\boldsymbol{x}_n)$ is now available after the consensus step.

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \propto \frac{\eta(\boldsymbol{x}_n)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}} \tag{2.18}$$

The prediction term $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$ in denominator can be calculated as follows,

$$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \int_{\mathbb{R}^d} f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1}) f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1}) d\boldsymbol{x}_{n-1}, \tag{2.19}$$

where $f(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})$ can be calculated according to the transition model while $f(\boldsymbol{x}_{n-1}|\boldsymbol{y}_{1:n-1})$ is the global posterior of the previous time instant and hence also available at each sensor. To recover the global posterior, importance sampling is again adopted. In order to make sure the sampling particles cover most support of the global posterior, we draw half of the particles from $\eta(\boldsymbol{x}_n)$ while the remaining from $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$. For the particles sampling from $\eta(\boldsymbol{x}_n)$, the importance weight is calculated according to

$$\omega_n^m = \frac{\eta(\boldsymbol{x}_n)^{K-1}}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}}. \tag{2.20}$$

And for particles sampling from $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})$, the importance weight is calculated according to

$$\omega_n^m = \frac{\eta(\boldsymbol{x}_n)^{K}}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K}}. \tag{2.21}$$

At last, a Gaussian mixture approximation of global posterior can be learned from the obtained weighted particles $\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M$. Here the total number of sensors also has to be known in advance.

The pseudo code of GM-DPF algorithm is presented in Algorithm 6.

Algorithm 4 Gaussian Mixture Learning(GML)

Require: $\{\boldsymbol{x}^m, \omega^m\}_{m=1}^M$
Initialize $C, \{\alpha_c, \boldsymbol{\mu}_c, \Sigma_c\}_{c=1}^C$
repeat
  for $m = 1, \cdots, M$ **do** ▷ E-step
    for $c = 1, \cdots, C$ **do**
      $p_{m,c} = \alpha_c \mathcal{N}(\boldsymbol{x}^m | \boldsymbol{\mu}_c, \Sigma_c)$
    end for
    normalize $\{p_{m,c}\}_{c=1}^C$
    for $c = 1, \cdots, C$ **do** ▷ M-step
      $\alpha_c = \sum_{m=1}^M p_{m,c}\omega_m$
      $\boldsymbol{\mu}_c = \alpha_c^{-1} \sum_{m=1}^M p_{m,c}\omega_m \boldsymbol{x}_m$
      $\Sigma_c = \alpha_c^{-1} \sum_{m=1}^M p_{m,c}\omega_m (\boldsymbol{x}^m - \boldsymbol{\mu}_c)(\boldsymbol{x}^m - \boldsymbol{\mu}_c)^T$
    end for
  end for
  normalize $\{\alpha_c\}_{c=1}^C$
until convergence
return $\{\alpha_c, \boldsymbol{\mu}_c, \Sigma_c\}_{c=1}^C$

Algorithm 5 Fine-tuning

Require: $\cup_{j \in \{k, \mathcal{N}_k\}} \{\alpha_c^j, \boldsymbol{\mu}_c^j, \Sigma_c^j\}_{c=1}^C$
Initialize D_k(the kth diagonal element of D)
for $c = 1, \cdots, C$ **do**
  for $j = 1, \cdots, D_k$ **do**
    calculate the KLD between $\{\alpha_c^k, \boldsymbol{\mu}_c^k, \Sigma_c^k\}$ and every left Gaussian component of node j
    Select the Gaussian component $\{\alpha_{c_j}^j, \boldsymbol{\mu}_{c_j}^j, \Sigma_{c_j}^j\}$ with minimum KLD
    Remove the selected Gaussian component
  end for
  $\{\tilde{\alpha}_c^{d_k}\}_{d_k=1}^{D_k+1} = \textbf{normalize}(\{\alpha_c^k, \{\alpha_{c_{d_k}}^{d_k}\}_{d_k=1}^{D_k}\})$
  $\boldsymbol{\mu}_c^{ft} = \sum_{d_k=1}^{D_k+1} \tilde{\alpha}_c^{d_k} \boldsymbol{\mu}_{c_{d_k}}^{d_k} (note: \boldsymbol{\mu}_{c_1}^1 = \boldsymbol{\mu}_c^k)$
  $\Sigma_c^{ft} = \sum_{d_k=1}^{D_k+1} (\Sigma_{c_{d_k}}^{d_k} + (\boldsymbol{\mu}_{c_{d_k}}^{d_k} - \boldsymbol{\mu}_c^{ft})(\boldsymbol{\mu}_{c_{d_k}}^{d_k} - \boldsymbol{\mu}_c^{ft})^T) \quad (note: \Sigma_{c_1}^1 = \Sigma_c^k)$
  $\alpha_c^{ft} = (\alpha_c^k + \sum_{d_k=1}^{D_k} \alpha_{c_{d_k}}^{d_k})/(D_k + 1)$
end for
return $\{\alpha_c^{ft}, \boldsymbol{\mu}_c^{ft}, \Sigma_c^{ft}\}_{c=1}^C$

Algorithm 6 GM-DPF

Require: $\{\boldsymbol{x}_{n-1,k}^m, \omega_{n-1,k}^m\}_{m=1,k=1}^{M,K}, \{\boldsymbol{y}_{n,k}\}_{k=1}^K$
Initialize M, L_b(iterations)
$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1,k}^m, \omega_{n-1,k}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}) \quad \forall k = 1, \cdots, K$
$\eta_k(\boldsymbol{x}_n) = \mathbf{GML}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M) \quad \forall k = 1, \cdots, K$
$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}) = \mathbf{GML}(\{\boldsymbol{x}_{n,k}^m, \omega_{n-1,k}^m\}_{m=1}^M) \quad \forall k = 1, \cdots, K$
for $i = 1, \cdots, L_b$ **do** ▷ Fusion step
  for $k = 1, \cdots, K$ **do**
    sensor k sends $\eta_k^i(\boldsymbol{x}_n)$ to sensor $j \in \mathcal{N}_k$
  end for
  for $k = 1, \cdots, K$ **do**
    $\eta_k^{i+1}(\boldsymbol{x}_n) = \prod_{j \in \mathcal{N}_k} (\eta_j^i(\boldsymbol{x}_n))^{\epsilon_{k,j}}$ ▷ average consensus
  end for
end for
for $i = 1, \cdots, L_b$ **do**
  for $k = 1, \cdots, K$ **do**
    sensor k sends $\eta_k^i(\boldsymbol{x}_n)$ to sensor $j \in \mathcal{N}_k$
  end for
  for $k = 1, \cdots, K$ **do**
    $\eta_k^{i+1}(\boldsymbol{x}_n) = \mathbf{Fine\text{-}tuning}(\eta_k^i(\boldsymbol{x}_n))$
  end for
end for
$f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \propto \frac{\eta(\boldsymbol{x}_n)^K}{f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1})^{K-1}} \quad \forall k = 1, \cdots, K$ ▷ Recovery step
Sample $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ from $f(\boldsymbol{x}_n|\boldsymbol{y}_{1:n}) \quad \forall k = 1, \cdots, K$
return $\{\boldsymbol{x}_{n,k}^m, 1/M\}_{m=1,k=1}^{M,K}$

2.5 Numerical Results

In this Section, we are going to test the above two DPFs under the same simulation environment to check their tracking performance. First, we will explore the effect of hyperparameters in each algorithm on the final estimate. The performance of the two algorithms will then be compared in terms of computation, communication complexity.

2.5.1 Performance metrics

To quantify the performance the proposed algorithm can achieve, we introduce an index called the *time-averaged root mean square error*(ARMSE).

Definition 2.3: [11] *ARMSE:*

$$\sqrt{\frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|_2^2}, \tag{2.22}$$

where N *denotes the time length while* $\boldsymbol{x}_n$ *and* $\hat{\boldsymbol{x}}_n$ *denote the true state and estimated state respectively.*

2.5.2 Graph Laplacian Distributed Particle Filtering

In this Section, we will show and analyze the simulation results of GL-DPF. Table 2.2 gives parameter settings we used in simulation.

Table 2.2: Simulation setup for GL-DPF

parameter	value
particle size	2000
k-NN parameter	5
gossip iterations	[50 100 200]
approximation parameter z	[100 500 1000]
Monte Carlo experiments	100x

Fig 2.3 and Fig 2.4 show the estimated trajectory and state vector of the target over time when the gossip iteration equals 100, approximation parameter z equals 500 respectively. The red dots and yellow dotted lines in Fig 2.3 represent the sensors and communication links. As you can see, the target was well-tracked.

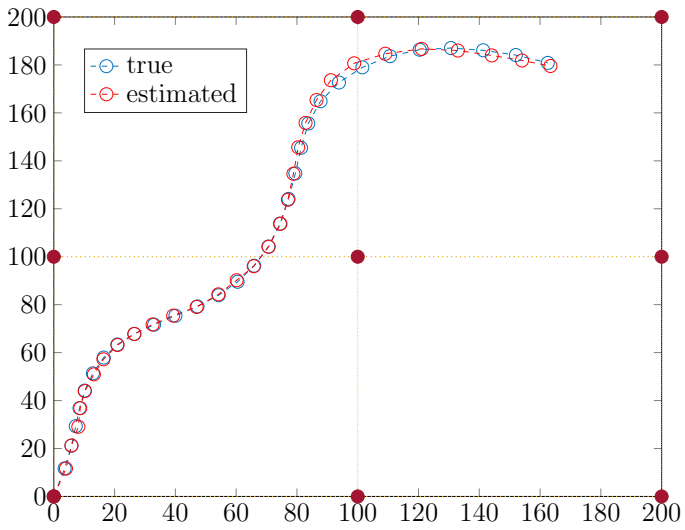


Figure 2.3: A target trajectory and the corresponding estimated trajectory obtained with GL-DPF.

Our goal is to reduce communication overhead while guaranteeing acceptable estimation performance. The communication overhead of the algorithm is determined by the number of gossip iterations as well as the graph Laplacian approximation parameter z. We need to play with those two parameters to see changes of the tracking performance, which is measured by the metric ARMSE and the result is shown in Fig 2.5.

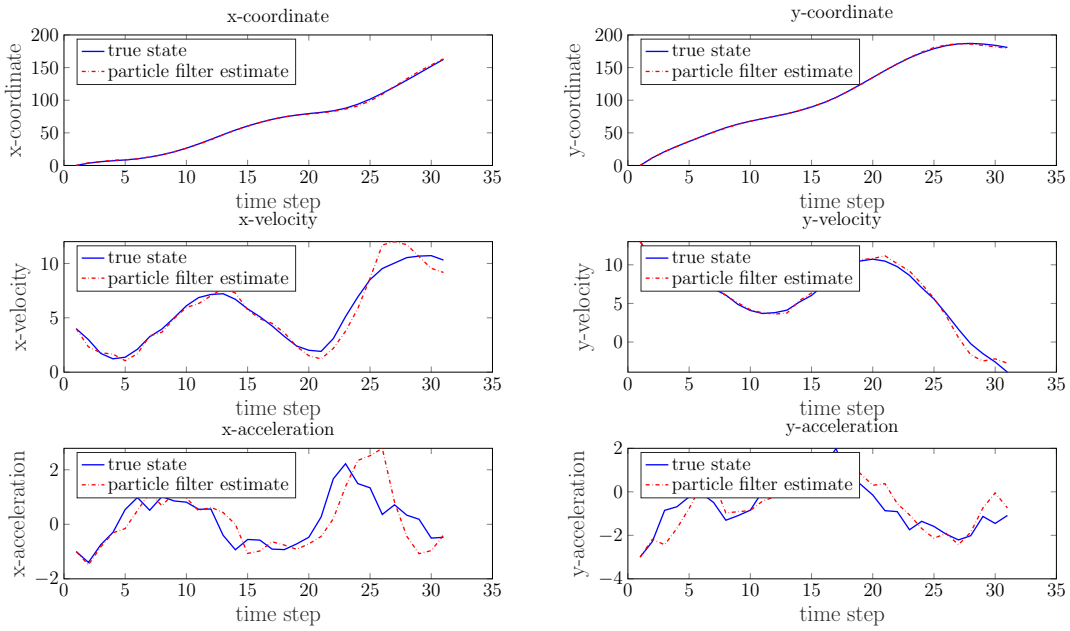


Figure 2.4: 6-D state tracking performance.

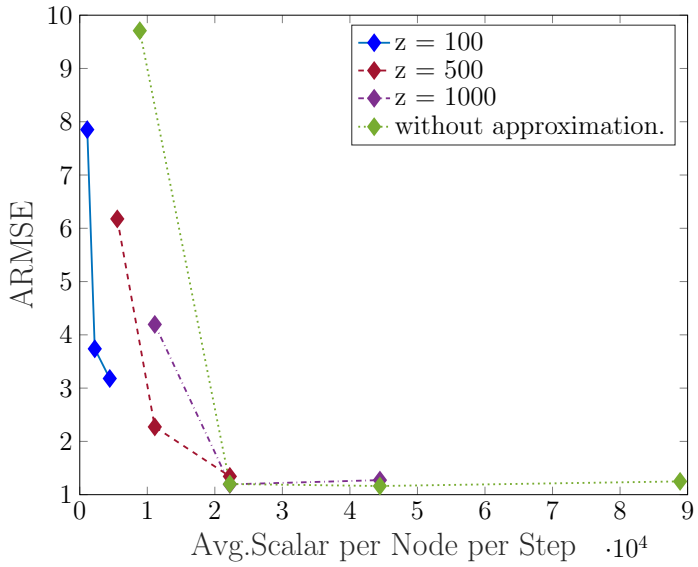


Figure 2.5: The value of ARMSE as a function of the average size of transmitted data per node per time step(the cross, diamond, circle and square marks correspond to 20, 50, 100, 200 randomized gossip iterations respectively).

In Fig 2.5, each curve corresponds to a fixed value of parameter z and the number of gossip iterations varies from 50 to 200(50, 100, and 200 in our case). What is worth mentioning is that the blue curve in the figure shows the estimating performance without applying the graph Laplacian approximation technique and serves as

the benchmark. Different from the other three curves, we add additionally one more choice of gossip iterations, i.e. 20 gossip iterations, as you can see from the figure that the blue curve has four inflection points while the other three only have three. Besides, for the 9-node sensor network, the averaging error is significantly small when 200 gossip iterations are implemented. From the figure, we can deduce that choosing parameter z and gossip iterations properly could lead to a big reduction in communication cost while keeping the ARMSE still relatively slow. And for all three choices of z presented in the figure, at least 100 gossip iterations are required. To sum up, if limited communication resource is provided, the combination of small z and sufficient gossip iterations would be a better choice. On the contrary, if better estimation performance always comes first, then we can appropriately increase the parameter z.

2.5.3 Gaussian Mixture Distributed Particle Filtering

In this Section, we will show and analyze the simulation results of GM-DPF. Table 2.3 gives parameter settings we used in simulation.

Table 2.3: Simulation setup for GM-DPF

parameter	value
particle size (local estimate)	2000
particle size(fusion)	5000
particle size(recovery)	5000
Gaussian components(C)	1(default)
EM iterations	100(default)
consensus iterations	[1 2 3 4 5]
fine-tuning iterations	[1 2 3 4 5]
Monte Carlo experiments	20x

To build a benchmark for GM-DPF, we implement the Gaussian mixture approximation in the centralized sense. In the centralized scenario, each sensor first performs the local particle filtering and GMM approximation. Next, all sensors send their GMM statistic of both prior and posterior to the fusion center. The fusion center performs the fine-tuning step on the prior distribution and a fusion and recovery step on the posterior information. Finally, it broadcasts the GMM statistic of the estimated global posterior to all sensors to proceed to the next time instant.

In Fig 2.6, we present the simulation result of ARMSE as a function of both numbers of consensus as well as fine-tuning iterations. Different from the randomized gossip algorithm we used in GL-DPF, here each sensor communicates with all its neighbors in each iteration. For comparison, we also implemented this algorithm in the centralized manner which can serve as the benchmark. From the figure we can find that the ARMSE decreases as the number of consensus iterations increases and the ARMSE gradually approaches that in the centralized case.

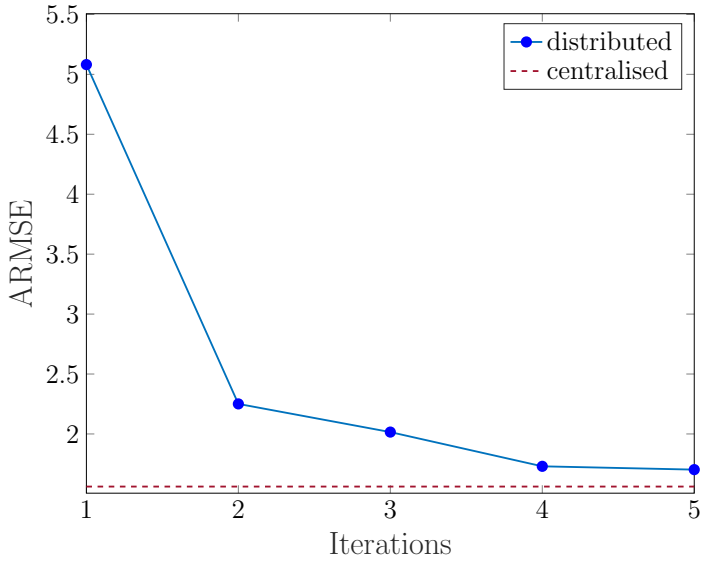


Figure 2.6: The value of ARMSE as a function of the number of consensus iterations.

In Section 2.4.1, we also introduce some techniques to help us determine the number of Gaussian components, and we apply the information criteria to our learning process of the Gaussian mixture model in the simulation to see the difference it brings. Since the choice of the number of Gaussian components is not influenced by the distributed setup, we decide to verify it under the centralized structure. The parameter settings are given in Table 2.4 when AIC and BIC are used to determine the number of Gaussian components. Compared to the Table 2.3, the number of EM iterations is larger to ensure the EM algorithm will converge. Simulation results in Table 2.5 show that performance improves when the number of Gaussian components is adaptively determined using the information criteria.

Table 2.4: Simulation setup of GM-DPF using information criteria

parameter	value
max Gaussian components	5
EM iterations	2000
consensus iterations	10
fine-tuning iterations	10
Monte Carlo experiments	20x

* the number of particles we used is consistent with that in Table 2.3.

Table 2.5: Improvements to ARMSE when information criteria are used

Information criterion	Absent	AIC	BIC
ARMSE	1.56	1.48	1.34

2.6 Summary

In this Chapter, extensive research on DPF is conducted, followed by the reproduction work of 2 state-of-the-art DPFs, GL-DPF and GM-DPF respectively. Next, we will briefly analyze and discuss the above algorithms.

2.6.1 Complexity Analysis

Table 2.6: Communication overhead and computational complexity analysis(Chapter 2) (M: the size of the particle set; N_{knn}: the number of nearest neighbors in k-NN; d: the dimension of the state space; L_g: the number of randomized gossip iterations in GL-DPF; L_b: the number of consensus iterations in GM-DPF; z: the number of reserved Laplacian transfer coefficients; K: the number of agents; C:the number of components in GMM; L_g: the number of EM iterations; L_f: the number of iterations for fusion stage; L_{ft}: the number of iterations for fine-tuning.)

Algorithm	GL-DPF	GM-DPF
communication	$\mathcal{O}(2L_g z/K)$	$\mathcal{O}(L_b C d^2) - \mathcal{O}(K L_b C d^2)$
computation	$\mathcal{O}(M^3 + N_{knn} M d)$	$\mathcal{O}([(L_g + K)L_f M + M + K L_{ft}] C d^2)$

- GL-DPF
  - Computation complexity
    The complexity of particle filtering is omitted here, and we only focus on the additional computation burden of the introduced approximation algorithm. The computation process of GL-DPF can be divided into 2 subtasks, Eigenvalue decomposition, and k-NN search. The complexity of the eigenvalue decomposition of an M by M matrix is $\mathcal{O}(M^3)$. And the complexity of the k-NN search algorithm is $\mathcal{O}(N_{knn} M d)$. Hence, the particle set size is limited to $\mathcal{O}(M^3 + N_{knn} M d)$.
  - Communication complexity
    In GL-DPF, the number of scalars transmitted across the network depends on the approximation parameter z, and the randomized gossip algorithm is used here to reach the consensus. In every gossip iteration, there are in total

$2z$ scalars transmitted on the chosen communication link since the information exchange process is bidirectional. Hence, the average communication complexity per agent during each time step is $\mathcal{O}(2L_g z/K)$.

- GM-DPF
  - Computation complexity
    The fusion part consists of three subtasks: namely local posterior learning (GMM learning), GM fusion, and GM recovery. For Gaussian mixture learning, it costs $\mathcal{O}(L_g MCd^2)$. In GM fusion, the complexity is $\mathcal{O}(KMCd^2)$, where $\mathcal{O}(K)$ means a sensor has at most K neighbors. In GM recovery, complexity is $\mathcal{O}(MCd^2)$. Besides, in fine-tuning step, the complexity is $\mathcal{O}(KCd^2)$. And we assume there are L_f and L_{ft} iterations for fusion and fine-tuning respectively. Thus the overall complexity is $\mathcal{O}([(L_g+K)L_f M+M+KL_{ft}]Cd^2)$.
  - Communication complexity
    In GM-DPF, Gaussian mixture statistics are transmitted across the network. Let C denote the number of components in GMM we assign, then $(d^2+d+1)C$ scalars need to be transmitted. Due to the symmetry of the covariance matrix, the number of transmitted scalars of the covariance matrix can be reduced to $(d^2+d)/2$. Besides, since all the component weights sum to one, we only need to transmit $C-1$ components number instead of C. Hence, the total number of scalars representing a Gaussian mixture is $Cd^2/2+(C/2+1)d+C-1)$. In a consensus iteration, every agent sends its Gaussian mixture statistics to its neighbors and hence each communication link is used twice. The total number of Gaussian mixtures transmitted during a consensus iteration is $2|E|$, where $|E|$ is the number of links in the network. Let L_b denote the number of consensus iterations and K denotes the number of agents, the average communication complexity per agent during each time step is $2|E|L_b(Cd^2/2+(C/2+1)d+C-1)/K$. Since $|E|$ ranges from $\mathcal{O}(K)$ to $\mathcal{O}(K^2)$ for a connected network, the communication complexity should between $\mathcal{O}(L_b Cd^2)$ and $\mathcal{O}(KL_b Cd^2)$.

2.6.2 Discussion

Both GL-DPF and GM-DPF are distributed particle filter algorithms proposed in the last five years. However, both algorithms have some challenges in optimizing their parameters. For example, in the GL-DPF algorithm, how to choose the size of N_{knn} in the k-NN search has not been fully studied. At the same time, in some state spaces, using k-NN to build a graph signal may not be the optimal choice. For example, when the observed quantities are extremely sensitive to certain dimensions in the state space, then the weight correlation between particles is not simply related to the Euclidean distance between particles. And in GM-DPF, the network should also be able to adaptively learn the optimal number of Gaussian components in the Gaussian mixture model.

Next, we analyze the limitations and challenges of the two algorithms. In implementing GL-DPF, we make a strong assumption that each agent must hold the same

set of particles at every time instant. This requires each agent to hold a synchronized random generator, which becomes impractical in large network architectures. If the agents are not synchronized at the beginning, the max-gossip protocol is required to make all agents agree on the common seed before each generation of a new particle set, which introduces additional communication overhead. For GM-DPF, the author fits the particle set in a parametric fashion, so that sufficient, yet possibly overabundant particles are required to reduce the variance of the approximated GMM. In addition, since the analytical solution of the geometric mean of the function cannot be given, the author uses importance sampling. Still, at a cost, it greatly increases the computation and memory resources. At the same time, the learning of the Gaussian mixture model using the EM algorithm is also computationally intense.

2.6.3 Conclusion

The highlights of this Chapter are listed below.

- The literature review of DPF is conducted. We have classified the distributed particle filter according to the topology of the communication and the type of data transmitted in the communication process, and then summarized some of the latest DPF algorithms, as shown in Table 2.1.
- We have reviewed and reproduced two state-of-the-art DPFs, GL-DPF and GM-DPF respectively. And the numerical results confirm the contribution of both two algorithms in reducing communication resources.
- A comparison of the computational and communication complexity of these two DPFs has be given from a theoretical point of view. Additionally, their limitations and challenges are also analyzed.

Gaussian Process Enhanced Distributed Particle Filtering

3

In the previous Chapter, we reproduced two recently proposed DPFs and in this Chapter, inspired by the work in [10], we explore a new kind of DPF, where the particles themselves served as exchange quantities. The difficulty of this idea is how to condense the knowledge of local agents into as few particles as possible, so as to reduce the communication overhead.

In Section 3.1, the motivation for direct particle exchange will be presented. In Section 3.2, we will briefly introduce the Gaussian process(GP) enhanced resampling algorithm proposed in [10], which makes it possible to use fewer particles for state estimation. In Section 3.3, based on the above resampling algorithm and taking the particles themselves as the exchange quantities, a fusion center required DPF is first proposed and then decentralized. In Section 3.4, instead of fully accepting the received particles, each agent further resamples the received particles based on its local measurements to improve the tracking performance, and DPFs based on the consensus and diffusive modes are proposed respectively. Simulations are carried out for each algorithm.

3.1 Motivation

To reduce the communication resources of DPFs, we have reproduced two state-of-the-art DPFs whose nature of transmitting quantities are weights and posteriors respectively. And based on the limited information known to our current investigation, little work has been done into the transmission of particles themselves as information carriers since the performance of PF is positively related to the number of particles we use. In addition, transmitting a single particle means transmitting as many scalars as the number of dimensions, which can lead to unbearable communication overhead if the state space dimension of the system is too high. However, with the gradual maturity of research on particle filters, more and more types of proposal distribution and resampling algorithms have been proposed [8][10][42][43][44][45], solving the problem of particle impoverishment very well. These pleasing processes enable the particle filter algorithm to get rid of the high dependence on the number of particles. In other words, even with a very limited number of particles, the diversity of particles can also be well maintained. The above research advance makes it possible to exchange particles directly in the distributed particle filter architecture. At the same time, direct particle exchange also has the following advantages. Compared with parameterized expression, particle exchange better inherits the Monte Carlo idea of particle filtering and has better flexibility in modeling distributions. Compared with the transmission of compressed measurement data, it also works better in terms of privacy-preserving since the measurement data usually contains some information about the agent itself

(such as location, measurement function, etc.) and hence not suitable for some very privacy-sensitive scenarios. The follow-up of this thesis will conduct in-depth research on the DPF algorithm based on direct particle exchange.

3.2 Gaussian Process Enhanced Resampling

In this Section, we will mainly introduce the Gaussian process enhanced resampling algorithm (GP-R) proposed in [10]. In Section 3.2.1, a brief introduction to Gaussian process will be given. In Section 3.2.2, the resampling algorithm is formally introduced. The effectiveness of the Gaussian process enhanced resampling algorithm has already been verified in [10] compared to the standard resampling algorithm under the constant velocity (CV) model. In Section 3.3, the performance improvement brought by GP-enhanced resampling algorithm will be shown in the DPF case.

3.2.1 Gaussian Process

Gaussian process is an extension of multivariate Gaussian distribution when the dimension goes to infinity. From the probability theory perspective, a Gaussian process should have the property that every finite collection of random variables follows a multivariate normal distribution. In general, we can view the Gaussian process as distributions over functions. The three most popular applications for GP are regression, classification, and dimensionality reduction [46], and in this thesis, we aim to exploit the use of Gaussian process regression(GPR). Generally, there are two ways of viewing GP, the weight space view and the function space view respectively [47], and we work from the latter perspective.

In GPR, we first model our problem as follows,

$$y = f(\boldsymbol{x}) + \varepsilon, \tag{3.1}$$

where ε is the noise following $\mathcal{N}(0, \sigma_n^2)$. Unlike other parametric regressors, GPR has no assumptions on the underlying structure(e.g. linear, quadratic) of function f. In GPR, we assume the signal to be predicted is also a random variable which follows a particular distribution and the variance of the guessed distribution reflects our uncertainty regarding the function. The uncertainty can be reduced by feeding more supervised data into the model. In GPR, we use the following notation to show that the function f is expressed as a GP.

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), \kappa(\boldsymbol{x}, \boldsymbol{x}')) \tag{3.2}$$

A GP is determined by two terms, *mean* and *covariance* function. The mean function $m(\boldsymbol{x})$ provides the expected function output at input $\boldsymbol{x}$. The covariance models the dependence between the function outputs at different inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. The covariance matrix is influenced by the choice of the *kernel* which is denoted by κ and the choice of *kernel* is based on the prior knowledge on the given data, such as smoothness or other patterns. One of the most popular kernels is the radial basis function(RBF), which is defined as follows,

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\lambda^2}) + \sigma_n^2 \delta(\boldsymbol{x}, \boldsymbol{x}'), \tag{3.3}$$

where $\delta(\cdot)$ is the Kronecker delta operator. The two adjustable hyperparameters are λ and σ_f. λ is called the length scale. We can see from the definition, if $\boldsymbol{x} \approx \boldsymbol{x}'$, then $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ approaches the maximum value, which means these two points are strongly correlated. And if $\boldsymbol{x}$ is far way from $\boldsymbol{x}'$, the variance becomes nearly zero, which means $\boldsymbol{x}$ has negligible effect on $\boldsymbol{x}'$. σ_f is the predefined maximum covariance of signals.

Although GP is defined in continuous space and has infinite dimensions, in practice, there are always finite measurements. Before making the prediction, we first need to do some preparation work. Let $\boldsymbol{X}$ be a matrix, and each row of it represents a new input point $\boldsymbol{x}_i, i = 1, ..., n$. The function value y_* at position $\boldsymbol{x}_*$ is the prediction object. The covariance information among all the inputs and $\boldsymbol{x}_*$ can be integrated into the following three matrices.

$$\mathcal{K}(\boldsymbol{X}, \boldsymbol{X}) = \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_n, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix} \tag{3.4}$$

$$\mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*) = \begin{bmatrix} \kappa(\boldsymbol{x}_*, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_n) \end{bmatrix} \tag{3.5}$$

$$\mathcal{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \tag{3.6}$$

Remind yourself that every finite collection of random variables follows a multivariate normal distribution in GP. We then have

$$\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \sim \mathcal{N}(\boldsymbol{0}, \begin{bmatrix} \mathcal{K}(\boldsymbol{X}, \boldsymbol{X}) & \mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)^T \\ \mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*) & \mathcal{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix}), \tag{3.7}$$

where $(\cdot)^T$ is the matrix transpose operator. After applying the Bayes' theorem, the conditional probability of the prediction value can be written as

$$p(y_*|\boldsymbol{y}) \sim \mathcal{N}(\mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)\mathcal{K}(\boldsymbol{X}, \boldsymbol{X})^{-1}\boldsymbol{y}, \mathcal{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) - \mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)\mathcal{K}(\boldsymbol{X}, \boldsymbol{X})^{-1}\mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)^T). \tag{3.8}$$

where the *mean* value $\mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)\mathcal{K}(\boldsymbol{X}, \boldsymbol{X})^{-1}\boldsymbol{y}$ is our best estimation for y_*, and the *variance* $\mathcal{K}(\boldsymbol{x}_*, \boldsymbol{x}_*) - \mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)\mathcal{K}(\boldsymbol{X}, \boldsymbol{X})^{-1}\mathcal{K}(\boldsymbol{X}, \boldsymbol{x}_*)^T$ shows our uncertainty on the estimation.

3.2.2 Gaussian Process Enhanced Resampling Algorithm

In this Section, the Gaussian process enhanced resampling algorithm [10] which uses the GPR to model the posterior will be presented. The advantage of using GPR is that in addition to its ability to provide the estimation(prediction) results, the uncertainty can also be measured in forms of variance. This extra information can also be utilized in the resampling stage and make it possible to use fewer particles to cover the information

in the state space. In a word, a better exploration-versus-exploitation trade-off can be achieved by deciding whether to sample from more accurate posterior features or exploiting the uncertainty of this distribution. Exploitation means to focus the search on a local space where a current good solution exists while exploration means to explore the search space on a global scale.

Before the resampling stage, GPR is firstly used to model the posterior which is represented by a set of weighted particles in PF. Given the pairs of particles and weights $\{\boldsymbol{x}^m, \omega^m\}_{m=1}^M$ where $\boldsymbol{x}^m$ corresponds to the m^{th} row in $\boldsymbol{X}$ while ω^m corresponds to the m^{th} entry in $\boldsymbol{\omega}$, the *mean* and *covariance* functions of the approximated GP(denoted as $\mathcal{GP}1$ in the pseudo-code) can be written as

$$\mu(\boldsymbol{x}) = \mathcal{K}(\boldsymbol{X}, \boldsymbol{x})\boldsymbol{\alpha}, \tag{3.9}$$

$$s^2(\boldsymbol{x}) = \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{\beta}^T\boldsymbol{\beta}, \tag{3.10}$$

where

$$\boldsymbol{\alpha} = \mathcal{K}(\boldsymbol{X}, \boldsymbol{X})^{-1}\boldsymbol{\omega} = (\boldsymbol{L}^T)^{-1}\boldsymbol{L}^{-1}\boldsymbol{\omega}, \tag{3.11}$$

$$\boldsymbol{\beta} = \boldsymbol{L}^{-1}\mathcal{K}(\boldsymbol{X}, \boldsymbol{x})^T. \tag{3.12}$$

$\boldsymbol{X}$ is the particle set following $\boldsymbol{X} = [\boldsymbol{x}^1; \cdots; \boldsymbol{x}^M]$, and $\boldsymbol{\omega}$ is the weight information corresponding to $\boldsymbol{X}$. To accelerate the matrix inverse operation, we here apply the Cholesky decomposition to $\mathcal{K}(\boldsymbol{X}, \boldsymbol{X})$, making $\mathcal{K}(\boldsymbol{X}, \boldsymbol{X}) = \boldsymbol{L}\boldsymbol{L}^T$. And the Gaussian *kernel* in (3.13) is used in our regression.

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(\frac{1}{2\sigma^2}||\boldsymbol{x} - \boldsymbol{x}'||^2) \tag{3.13}$$

Since the estimation results are the particles' weights, the normalization step (3.14) has to be performed afterward.

$$p(\boldsymbol{x}) = \frac{\mu(\boldsymbol{x})}{\int \mu(\boldsymbol{x})d\boldsymbol{x}} = \frac{\mu(\boldsymbol{x})}{\sum_{m=1}^M \alpha_m \int \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^m)d\boldsymbol{x}} \tag{3.14}$$

To exploit the uncertainty knowledge provided by GPR and enable the exploration round, we artificially build a second particle set using the function $f(\boldsymbol{x}) = \mu(\boldsymbol{x}) + 3s(\boldsymbol{x})$. According to the empirical rule, 99.7% of the values lie within three standard deviations of the mean [48]. Hence, the information provided by the variance term is fully exploited. The input set for the function is generated by selecting $M-1$ intermediary particles from the existing ones. By doing so, the points with high variances are also taken into consideration. The second particle set can be denoted as $\{\hat{\boldsymbol{x}}^m, f(\hat{\boldsymbol{x}}^m)\}_{m=1}^{M-1}$ and then approximated by another GP model(denoted as $\mathcal{GP}2$ in the pseudo-code).

To perform the resampling, we need to resort to some strategy to efficiently sample from the GP. Due to the choice of the Gaussian *kernel*, sampling from a GP is equivalent to sampling from a GMM. To see this, we can first rewrite (3.14) as

$$p(\boldsymbol{x}) = \sum_{m=1}^{M} \tilde{\alpha}_m \kappa(\boldsymbol{x}, \boldsymbol{x}^m), \tag{3.15}$$

where $\tilde{\alpha}_m = \alpha_m / \int \mu(\boldsymbol{x}^m) d\boldsymbol{x}$, and $\int \mu(\boldsymbol{x}) d\boldsymbol{x} = (\sigma\sqrt{2\pi})^d \sum_{m=1}^{M} \alpha_m$. d is the dimension of particles. Now it is easy to see that (3.15) is equivalent to a GMM with mixture weight $\sqrt{2\pi}\sigma\tilde{\alpha}_m$. Analogously to sampling from the first GP, the same operations should again be applied to the second GP to complete the whole resampling stage. To decide whether to sample from $\mathcal{GP}1$ or $\mathcal{GP}2$, we can first specify a parameter ζ and generate a scalar u from a uniform distribution between 0 and 1, if $u < \zeta$, we sample from $\mathcal{GP}2$, otherwise from $\mathcal{GP}1$. By tuning the parameter ζ, we can decide whether the final resampling results are more inclined to exploitation or exploration. The GP-enhanced resampling algorithm is presented in Algorithm 7.

Algorithm 7 Gaussian process enhanced resampling(GP-R)

Require: $\{\boldsymbol{x}_n^m, \omega_n^m\}_{m=1}^M, \zeta$
Initialize $M, \{l, \sigma\}$(GP hyperparameters)
Fit a GP $\mathcal{GP}1$ using particle set $\{\boldsymbol{x}_n^m, \omega_{n-1}^m\}_{m=1}^M$
for $m = 1, \cdots, M-1$ **do**
    $\hat{\boldsymbol{x}}_n^m = (\boldsymbol{x}_n^{m+1} - \boldsymbol{x}_n^m)/2 + \boldsymbol{x}_n^m$
    $\{\mu(\hat{\boldsymbol{x}}_n^m), s^2(\hat{\boldsymbol{x}}_n^m)\} = \mathcal{GP}1(\hat{\boldsymbol{x}}_n^m)$
    Compute $f(\hat{\boldsymbol{x}}_n^m) = \mu(\hat{\boldsymbol{x}}_n^m) + 3s(\hat{\boldsymbol{x}}_n^m)$
end for
Fit a GP $\mathcal{GP}2$ using particle set $\{\hat{\boldsymbol{x}}_n^m, f(\hat{\boldsymbol{x}}_n^m)\}_{m=1}^{M-1}$
for $m = 1, \cdots, M$ **do**
    Draw $u \sim \mathcal{U}_{[0,1]}$
    if $u < \zeta$ **then**
        Draw $\tilde{\boldsymbol{x}}_n^m$ from $\mathcal{GP}2$
    else
        Draw $\tilde{\boldsymbol{x}}_n^m$ from $\mathcal{GP}1$
    end if
end for
return $\{\tilde{\boldsymbol{x}}_n^m\}_{m=1}^M$

3.3 Direct Particle Exchange based Distributed Particle Filtering

In the previous Section, we introduced the Gaussian process enhanced resampling algorithm, which enables particle filtering to achieve good estimation results under a limited number of particles. In this Section, we investigate DPF based on direct particle exchange. In Section 3.3.1, we conduct the research in a fusion center(FC)-based network, followed by decentralization in Section 3.3.2. The simulation results are given. As a reminder, our goal throughout the thesis is to design an algorithm that allows each agent to approximate the global posterior probability $p(\boldsymbol{x}_n|\boldsymbol{y}_n)$ at each time instant n. And at time n, our known information includes the estimate of the global posterior probability by each agent k at the previous time, denoted as $\{\boldsymbol{x}_{n-1,k}^m, \omega_{n-1,k}^m\}_{m=1,k=1}^{M,K}$, and the measurement information of the whole network, denoted as $\{\boldsymbol{y}_{n,k}\}_{k=1}^K$.

3.3.1 FC-based Network

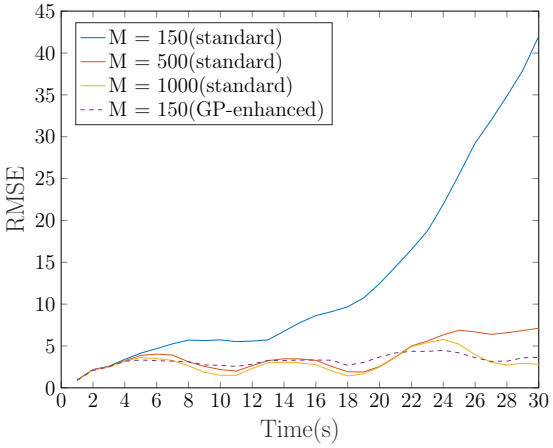
First, we consider there is a fusion center, which is responsible for particle "redistribution". The algorithm can be divided into 2 stages: local filtering and information fusion. In the first stage, all agents perform local PF with the GP-enhanced resampling algorithm after obtaining local measurements, and finally get a set of weighted particles(e.g., agent k would hold the particle set $\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M$). In the second stage, every agent sends a subset of particles (the size is determined by the pre-defined parameter $\phi \in (0, 1]$) to the fusion center, which finally ends up with a particle set, denoted as $\cup_{j=1}^K \{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}$. To fuse knowledge, M particles will be randomly picked at the fusion center and broadcast to each agent. The algorithm is presented in Algorithm 8.

Algorithm 8 GP-DPF(FC-based)

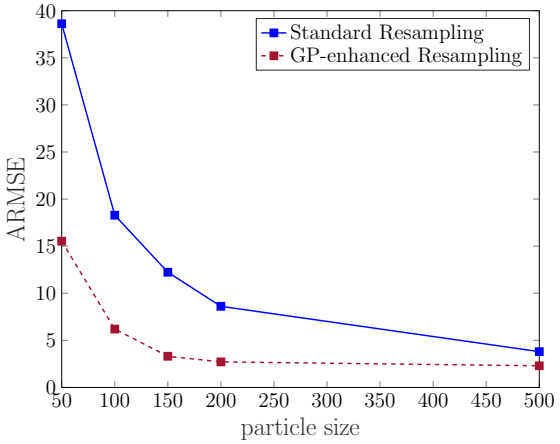
Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K, \phi \in (0, 1]$,
Execute at all nodes $k = 1, \cdots, K$ in parallel:
1: **Initialize** M
2: $\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$
3: $\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{GP\text{-}R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M)$
4: Randomly draw ϕM particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$
5: Sensor k sends $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$ to central unit
6: Central unit randomly draw M particles from $\cup_{j=1}^K \{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}$
7: Central unit broadcasts $\{\boldsymbol{x}_n^m\}_{m=1}^M$
8: $\boldsymbol{x}_n = mean(\{\boldsymbol{x}_n^m\}_{m=1}^M)$
9: **return** $\boldsymbol{x}_n, \{\boldsymbol{x}_n^m\}_{m=1}^M$

Next, we briefly analyze the advantages and disadvantages of this algorithm. First, since the GP-enhanced resampling algorithm is used to replace the standard resampling algorithm, fewer particles are required to maintain the diversity of the particle set. Second, because the algorithm is based on the direct exchange of the particles, there is no need for a priori strong assumptions (such as synchronization of particle generation in GL-DPF), making it easier to deploy in practical applications. The shortcomings of the algorithm are also obvious. First, the existence of FC greatly reduces the scalability and robustness of the network. Second, the algorithm is not carefully designed during the information fusion stage. Since the operation is only to randomly select a part of particles from each agent to form the final particle set, the performance may be relatively poor. For the first point, we will use the local broadcast communication protocol to achieve the decentralization which would be shown in Section 3.3.2. For the second point, several ideas are proposed in both Section 3.4 and Section 4.

200 Monte Carlo trials have been run and Fig 3.1 shows the simulation results of Algorithm 8. As a comparison, the simulation results of using the standard resampling algorithm (systematic resampling in [7]) are also presented. In the initialization stage, the parameter ϕ is set to $1/K$ to minimize communication overhead. However, in practical scenarios, since the number of agents in the network may be unknown, redundancy should be introduced when initializing ϕ(ϕ should be greater than $1/K$ to ensure the diversity of particles). Fig 3.1(a) shows the time-dependent tracking error for different



(a) Comparison of standard resampling and GP-enhanced resampling in terms of tracking error with time(RMSE)



(b) The value of ARMSE as a function of particle size.

Figure 3.1: Simulation results of Algorithm 8: GP-DPF(FC-based)

particle numbers. We can see that after using the GP-enhanced resampling algorithm, 150 particles are already enough to keep the tracking error around 3. However, if the standard resampling algorithm is used, under the same conditions, the tracking performance diverges over time. When the number of particles increases to 500, the error begins to converge. It is only when the number of particles increases to 1000 that the tracking performance is comparable to the GP-enhanced resampling algorithm using 150 particles. This result confirms the effectiveness of the GP-enhanced resampling algorithm under the condition of a limited number of particles and this is achieved by introducing the second particle set when fitting the Gaussian process, resulting in a better exploration-versus-exploitation trade-off. Fig 3.1(b) shows the ARMSE as a function of particle set sizes, indicating that the ARMSE converges faster and always has a better tracking performance when GP-enhanced resampling is used.

3.3.2 Fully Distributed Network

In this Section, we decentralize the GP-DPF(FC-based) proposed in the previous Section by enabling each agent to communicate only with its neighbors. In the distributed particle filter based on the direct exchange of particles, in order to achieve the purpose of information fusion of the whole network, we need to make the particle set of each agent fully mixed and this can be achieved with the guidance of the diffusion strategies. Later, in Section 3.4.2, the idea of diffusion strategies is also used. In order to distinguish it from diffusion strategies, we call it diffusive mode. Under diffusive mode, all agents communicate with their neighbors in each iteration to mix their particle sets. If multiple communication iterations are performed in a single time instant, each agent can gradually gain information from the entire network. In this Section, we carefully design the communication protocol to minimize the communication overhead.

Compared with Algorithm 8, we perform L times local broadcast communications

to fuse the information. At the same time, the degree matrix $D \in \mathbb{R}^{K\times K}$ of multi-agent network is used to determine the local particle size. Before performing local particle filtering and resampling, each agent first uses the local degree information to calculate the number of particles it needs to sample using the following equation

$$S_k = \lceil M/(D_k + 1) \rceil, \tag{3.16}$$

which is based on the premise that each agent's final set of particles is uniformly derived from itself and connected agents. $\lceil \cdot \rceil$ is the ceiling function and D_k is the degree of agent k. To ensure that the particle set received by each agent meets the required number of particles, each agent broadcasts the required number of particles to its neighbors again. Finally, the number of particles to be broadcast at agent k is determined by $R_k = \max_{j\in\{\mathcal{N}_k,k\}}\{S_j\}$. In the first iteration, each agent performs particle filtering and GP-enhanced resampling according to the previously calculated R_k and broadcasts the particle set to the connected agents.

After the communication process, each agent randomly reserves M particles from the received particle set as the initial particle set for the next time instant ($L_b = 0$) or the next communication iteration ($L_b > 0$). If multiple communication iterations are performed in a single time instant, in subsequent iterations, each agent no longer performs local particle filtering, and the GP-enhanced resampling is replaced by random sampling. The flow of this algorithm is shown in Algorithm 9.

Algorithm 9 GP-DPF(fully distributed)

Require: $\{\boldsymbol{x}^m_{n-1,k}, \omega^m_{n-1,k}\}^{M,K}_{m=1,k=1}, \{\boldsymbol{y}_{n,k}\}^K_{k=1}, \phi \in (0, 1]$

Execute at all nodes $k = 1, \cdots, K$ in parallel:

Initialize M, D, L_b(iterations)
Determine $S_k = \lceil M/(D_k + 1) \rceil$ and send it to neighbors
Determine $R_k = \max_{j\in\{\mathcal{N}_k,k\}}\{S_j\}$
$\{\boldsymbol{x}^m_{n,k}, \omega^m_{n,k}\}^M_{m=1} = \mathbf{PF}(\{\boldsymbol{x}^m_{n-1}, \omega^m_{n-1}\}^M_{m=1}, \boldsymbol{y}_{n,k})$
$\{\boldsymbol{x}^m_{n,k}, \omega^m_{n,k}\}^{R_k}_{m=1} = \mathbf{GP\text{-}R}(\{\boldsymbol{x}^m_{n,k}, \omega^m_{n,k}\}^M_{m=1})$
Sensor k sends $\{\boldsymbol{x}^m_{n,k}\}^{R_k}_{m=1}$ to its neighbors
Sensor k randomly draw $\{\boldsymbol{x}^m_{n,k}\}^M_{m=1}$ from $\cup_{j\in\{\mathcal{N}_k,k\}}\{\boldsymbol{x}^m_{n,j}\}^{R_j}_{m=1}$
for $i = 1, \cdots, L_b$ **do**
    Sensor k randomly keep R_k particles
    Sensor k sends $\{\boldsymbol{x}^m_{n,k}\}^{R_k}_{m=1}$ to its neighbors
    Sensor k randomly draw $\{\boldsymbol{x}^m_{n,k}\}^M_{m=1}$ from $\cup_{j\in\{\mathcal{N}_k,k\}}\{\boldsymbol{x}^m_{n,j}\}^{R_j}_{m=1}$
end for
$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}^m_{n,k}\}^M_{m=1})$
return $\{\boldsymbol{x}_{n,k}\}^K_{k=1}, \{\boldsymbol{x}^m_{n,k}\}^{M,K}_{m=1,k=1}$

500 Monte Carlo trials have been run and Fig 3.2 shows the simulation results of Algorithm 9, focusing on the change of ARMSE with an increasing number of iterations when $M = 50, 100, 200$, respectively. From the figure we can see that there is a significant decrease in ARMSE as the particle set size increases. We can also see that at the first several iterations, ARMSE also decreases with the increase of the number

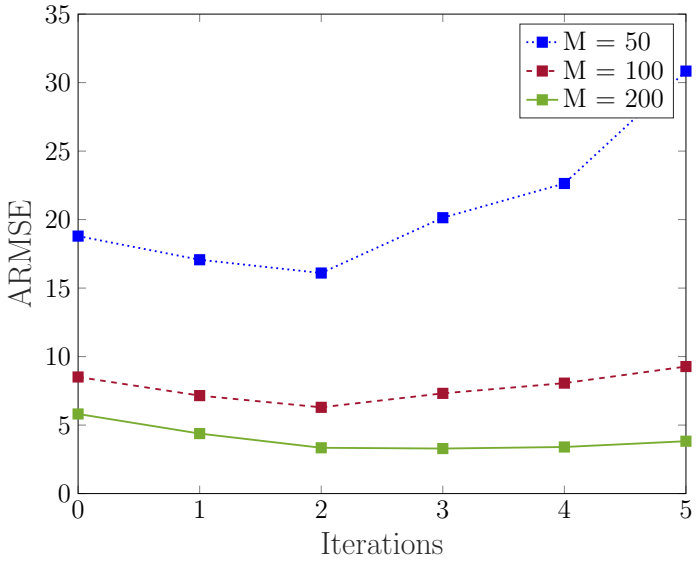


Figure 3.2: Simulation results of Algorithm 9, the value of ARMSE as a function of iterations.

of iterations since the information of the distant agents will be transmitted to the local in a multi-hop manner. However, we find that when the number of iterations is around 2 or 3, ARMSE begins to diverge. This is because each agent randomly keeps M particles from the received and discards the excess in each iteration, making the diversity of particles decrease after each iteration.

To avoid the divergence, two solutions are proposed here. The first is to increase the size of the particle set as much as possible. Taking the green line in the figure as an example, increasing the particle size can delay particle impoverishment and improve the degree of information fusion in the network. But at the cost, the communication and computing overhead will increase. The second solution is to control the number of iterations, such as setting the number of iterations to the diameter of the network.

3.4 Improved Direct Particle Exchange based Distributed Particle Filtering

In Section 3.3, direct particle exchange based DPF was proposed. Although the use of the GP-enhanced resampling algorithm reduces the dependence of the particle filter on the number of particles, the information fusion stage is not carefully designed, making the estimation performance unsatisfactory. In this Section, we will use the measurement information to filter out the particles that are the most representative or closest to the true state, thereby improving the performance.

In Section 3.4.1, with the help of the average consensus algorithm, each agent has access to information in the entire network at every moment. In Section 3.4.2, ideas similar to Section 3.3.2 are adopted. Each agent only communicates with neighboring agents, and the agent's information continues to diffuse to every corner of the network over time. The advantage of doing so is that the huge communication overhead brought

by the consensus algorithm can be avoided.

3.4.1 Consensus-based Fusion

In this Section, we adopt an idea of having each agent in the network score each particle generated in the network based on its local measurements. Then the sum of the scores of each particle can be used as the criterion for filtering out the most representative particles. Next, we briefly describe the flow of the algorithm.

The algorithm can be roughly divided into two stages. The goal of the first stage is to make sure that each agent can obtain the particles of the whole network, and the goal of the second stage is to calculate the global score(sum of scores from all agents) for each particle. The above goals are achieved through average consensus, and randomized gossip is used in our algorithm. Like the DPF algorithms in the previous Section, each agent first performs local particle filtering and GP-enhanced resampling to obtain a weighted particle set. After that, we randomly choose two connected agents and perform particle exchange and averaging. For example, after the first iteration, the particle set at connected agents i and j are as follows,

$$[\mathbf{0}\cdots\{\frac{x_{n,i}^{m,1}}{2}\frac{x_{n,i}^{m,2}}{2}\cdots\frac{x_{n,i}^{m,d}}{2}\}_{m=1}^{\phi M}\cdots\mathbf{0}\cdots\{\frac{x_{n,j}^{m,1}}{2}\frac{x_{n,j}^{m,2}}{2}\cdots\frac{x_{n,j}^{m,d}}{2}\}_{m=1}^{\phi M}\cdots\mathbf{0}], \tag{3.17}$$

where ϕM is the size of the local particle set and ϕ is the same as that in Algorithm 8. After sufficient iterations, every agent would have access to all particles, albeit with a scaling scalar K. Till now, the first stage has finished and there are in total ϕMK particles in each agent, denoted as $\{\boldsymbol{x}_n^m\}_{m=1}^{\phi MK}$. Based on the local measurements, every agent should calculate the score for each received particle and we use the likelihood value as the score which is defined as follows,

$$s_n^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_n^m). \tag{3.18}$$

The second average consensus round begins after the calculation procedure finishes. Same as the previous round, in every iteration, each agent selects one of its connected neighbors and performs the weight exchange and averaging. In the end, each agent would hold the following score set integrating all information from the network if sufficient iterations are conducted.

$$\{s_n^m\}_{m=1}^{\phi MK} = \{\frac{1}{K}\sum_{k=1}^{K} s_{n,k}^m\}_{m=1}^{\phi MK} \tag{3.19}$$

Finally, the resampling will be performed locally and M particles will be kept. The particle with a higher global score would be more likely to be chosen.

The algorithm is shown in Algorithm 10. The two additional parameters L_p and L_ω in initialization step represent the number of gossip iterations in the first and second average consensus rounds, respectively.

Algorithm 10 Improved GP-DPF(consensus-based)

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K, \phi \in (0,1]$

Initialize M, L_p, L_ω

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}) \quad \forall k = 1, \cdots, K$

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{GP\text{-}R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M) \quad \forall k = 1, \cdots, K$

Randomly draw ϕM particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M \quad \forall k = 1, \cdots, K$

for $i = 1, \cdots, L_p$ **do**

  Randomly choose 2 adjacent sensors k, j

  $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi MK} = \{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi MK} = \{\frac{1}{2}(\boldsymbol{x}_{n,j}^m + \boldsymbol{x}_{n,k}^m)\}_{m=1}^{\phi MK}$ ▷ average consensus

end for

$\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi MK} = K \times \{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi MK} \quad \forall k = 1, \cdots, K$

Compute particle score $\{s_{n,k}^m\}_{m=1}^{\phi MK} \quad \forall k = 1, \cdots, K$

for $i = 1, \cdots, L_\omega$ **do**

  Randomly choose 2 adjacent sensors k, j

  $\{s_{n,k}^m\}_{m=1}^{\phi MK} = \{s_{n,j}^m\}_{m=1}^{\phi MK} = \{\frac{1}{2}(s_{n,j}^m + s_{n,k}^m)\}_{m=1}^{\phi MK}$ ▷ average consensus

end for

$\{s_{n,k}^m\}_{m=1}^{\phi MK} = K \times \{s_{n,k}^m\}_{m=1}^{\phi MK}$

$\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \mathbf{Resample}(\{\boldsymbol{x}_{n,k}^m, s_{n,k}^m\}_{m=1}^{\phi MK}) \quad \forall k = 1, \cdots, K$

$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M) \quad \forall k = 1, \cdots, K$

return $\{\boldsymbol{x}_{n,k}\}_{k=1}^K, \{\boldsymbol{x}_{n,k}^m\}_{m=1,k=1}^{M,K}$

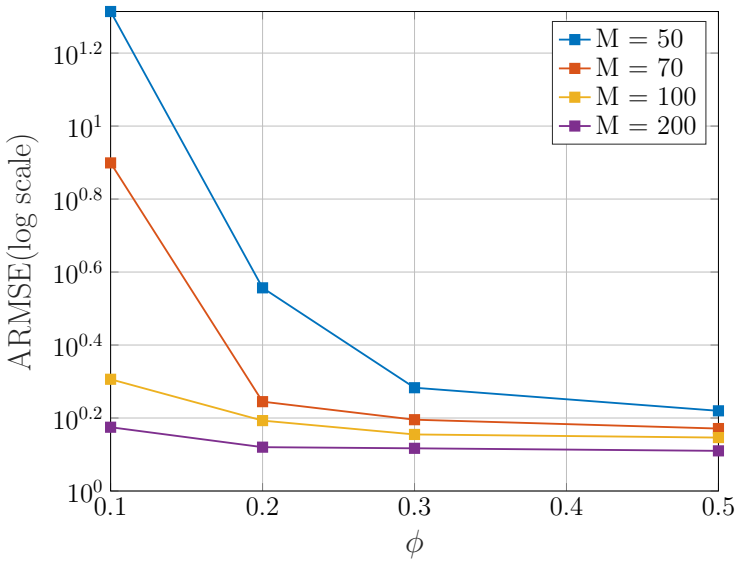
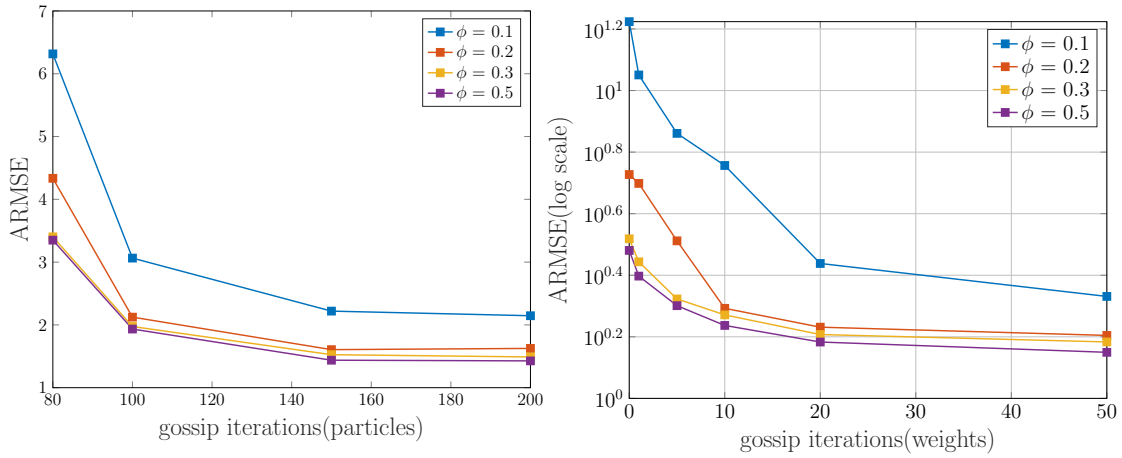


Figure 3.3: Simulation results of Algorithm 10(the value of ARMSE as a function of M and ϕ).

50 Monte trials have been run and the simulation results are presented in Fig 3.3 and 3.4. Compared with Section 3.3, the tracking error has been greatly reduced. To plot Fig 3.3, we ensure both L_p and L_ω are sufficiently large. It is shown that the ARMSE



(a) ARMSE as a function of ϕ and L_p(linear).

(b) ARMSE as a function of ϕ and L_ω(log).

Figure 3.4: Simulation results of Algorithm 10(M = 100).

decreases as the parameter ϕ increases due to the reason that particles in state space can be sufficiently explored. When we increase the local particle size M, the ARMSE would also decrease. Except for the reason that more particles can be exploited, another reason is that the local sensor would be able to learn a more convincing and reliable GP model if more data is fed.

In Fig 3.4(a), we present how ARMSE varies with an increasing number of gossip in the first consensus round. We have tested our algorithm under 4 different choices of parameter ϕ. Fig 3.4(a) shows that the ARMSE decreases as the number of gossip iterations increases, and it implies the particles recovered at sensors are more similar to the real exchanged particles and the consensus is reached at around 150 gossip iterations. Besides, choosing relatively larger ϕ can acquire much better tracking performance even though the average consensus has not been reached. To make the indicator, ARMSE, converge, sufficient gossip iterations have to be performed. Note, to run the simulation, we fix one of the iterations large enough and vary the other to see the changes of ARMSE. How ARMSE varies with an increasing number of gossip iterations in the second consensus stage is presented in Fig 3.4(b).

In some situations, the final estimation result may be very sensitive to some dimensions of the state space. For example, slight changes in acceleration will lead to great differences in the trajectory of the target. Hence, it is necessary to iterate as many times as possible in the first consensus round to ensure that the particles finally restored at each agent are consistent with the original.

3.4.2 Fusion under Diffusive Mode

In this Section, we combine the ideas of Algorithm 9 and Algorithm 10, and make some compromises. We don't just perform the task of particle exchange as in Algorithm 9, nor do we require each sensor to hold all the particles in the network as in Algorithm 10. Local broadcast communication is again adopted.

Same as Algorithm 9, in the first communication iteration, each agent performs local particle filtering as well as the GP-enhanced resampling, followed by broadcasting a subset of particles to its neighbors. However, here agents no longer keeps M particles by random. Based on its local measurement, each agent calculates the scores of the received particles and uses scores as the criteria for selection. As you can see from Algorithm 11, in the final particle set composition, take agent k as an example, $M/(D_k+1)$ particles are from the local particle set(i.e., $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$), and the remaining $MD_k/(D_k+1)$ particles are from the particles received from connected agents. The purpose of sampling separately is to make the final particles come from each agent evenly, to better increase the diversity of particles. Although it does not integrate the knowledge of all measurements when calculating the score of each particle, the knowledge of two agents is fused in each iteration. Over time, or by performing multiple iterations in each time instant, information will be diffused across the network. Parameters such as local particle size and the number of iterations can be adjusted to seek a balance between tracking performance and communication overhead. The algorithm is shown in Algorithm 11 and the simulation result is presented in Fig 3.5.

Algorithm 11 Improved GP-DPF(diffusive mode)

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K, \phi \in (0,1]$

Execute at all nodes $k = 1, \cdots, K$ in parallel:

Initialize M, L_b(iterations), D

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{GP\text{-}R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M)$

Randomly draw ϕM particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

Sensor k sends $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$ to its neighbors

Randomly draw $M/(D_k+1)$ particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$(the first part of particle set)

Compute scores $\{s_{n,k}^m\}_{m=1}^{\phi M D_k}$ corresponding to $\cup_{j\in\mathcal{N}_k}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}$

$\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{MD_k/(D_k+1)} = \mathbf{Resample}(\cup_{j\in\mathcal{N}_k}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}, \{s_{n,k}^m\}_{m=1}^{\phi M D_k})$(the second part of particle set)

Combine the two subsets of particles and get $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

for $i = 1, \cdots, L_b$ **do**

    Randomly draw ϕM particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

    Sensor k sends $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$ to its neighbors

    Randomly draw $M/(D_k+1)$ particles from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{\phi M}$(the first part of particle set)

    Compute scores $\{s_{n,k}^m\}_{m=1}^{\phi M D_k}$ corresponding to $\cup_{j\in\mathcal{N}_k}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}$

    $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^{MD_k/(D_k+1)} = \mathbf{Resample}(\cup_{j\in\mathcal{N}_k}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^{\phi M}, \{s_{n,k}^m\}_{m=1}^{\phi M D_k})$(the second part of particle set)

    Combine the two subsets of particles and get $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

end for

$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M)$

return $\{\boldsymbol{x}_{n,k}\}_{k=1}^K, \{\boldsymbol{x}_{n,k}^m\}_{m=1,k=1}^{M,K}$

20 Monte Carlo trials have been run and in simulations, we investigate the tracking

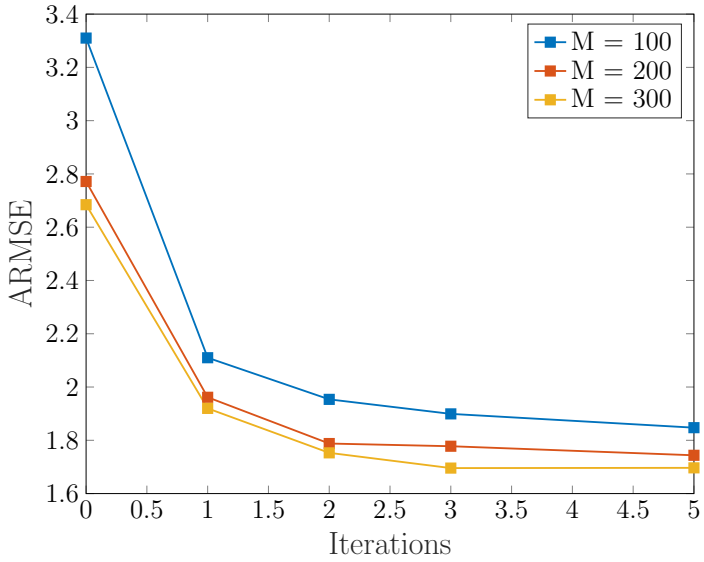


Figure 3.5: Simulation results of Algorithm 11(ARMSE as a function of local particle size and the number of iterations($\phi = 0.5$)).

error as a function of the number of iterations for three different-sized particle sets, 100, 200, and 300 respectively. We can see from Fig 3.5 that increasing the particle set size or increasing the number of iterations can both improve the tracking performance of the algorithm. At the same time, when the number of iterations reaches around 3, the tracking error begins to gradually converge. To sum up, when the local measurements are introduced to screen particles, the tracking performance is greatly improved.

3.5 Summary

In this Section, we will briefly analyze and discuss the GP-DPF algorithms proposed in this Chapter.

3.5.1 Complexity Analysis

Table 3.1: Communication overhead and computational complexity analysis (Chapter 3) (M: the size of the particle set; d: the dimension of the state space; D_{min}: the minimum degree of the given multi-agent network; L_b: the number of broadcast iterations; ϕ: a scalar between (0,1]; K: the number of agents; L_p: the number of gossip iterations in the first consensus round in Algorithm 9; L_ω: the number of gossip iterations in the second consensus round in Algorithm 9.)

Algorithm	Algorithm 9	Algorithm 10	Algorithm 11
communication	$2dML_b/(D_{min}+1)-$ $2KdML_b/(D_{min}+1)$	$\mathcal{O}(2L_p\phi Md+2L_\omega\phi M)$	$\mathcal{O}(2\phi dML_b)$ – $\mathcal{O}(2\phi dML_bK)$
computation	$\mathcal{O}(M^3+(M-1)^3)$	$\mathcal{O}(M^3+(M-1)^3)$	$\mathcal{O}(M^3+(M-1)^3)$

- Algorithm 9
  In Algorithm 9, particles are transmitted across the network. Since the number of particles to be transmitted is determined by the degree information of the network, let D_{min} denotes the minimum degree of the given network, and the maximum particle set to be transmitted by every agent is bounded by $M/(D_{min}+1)$. Let L_b denotes the number of broadcast iterations. There are $2|E|dM/(D_{min}+1)$ scalars that need to be transmitted in every iteration since the total number of communication rounds during an iteration is $2|E|$. Finally, the average communication complexity per agent during each time step is limited between $2dML_b/(D_{min}+1)$ and $2KdML_b/(D_{min}+1)$.

- Algorithm 10
  In Algorithm 9, particles, as well as the corresponding "weights", are transmitted across the network and the randomized gossip is used to achieve the consensus. In the first consensus round, there are $2\phi MdK$ scalars that need to be transmitted in every gossip iteration. Let L_p denotes the number of gossip iterations in the first consensus round, the communication overhead turns to be $2L_p\phi MdK$. In the second consensus round, there are $2\phi MK$ scalars that need to be transmitted in every gossip iteration. Let L_ω denotes the number of gossip iterations in the second consensus round, the communication overhead then turns to be $2L_p\phi MK$. Finally, the average communication complexity per agent during each time step is $\mathcal{O}(2L_p\phi Md+2L_\omega\phi M)$.

- Algorithm 11

In Algorithm 11, the randomized gossip algorithm Algorithm 9 is replaced by local broadcast communication and there is also no need for "weights" exchange. Let L_b denotes the number of broadcast iterations. There are $2|E|\phi dM$ scalars that need to be transmitted in every iteration. Finally, the average communication complexity per agent during each time step should between $\mathcal{O}(2\phi dML_b)$ and $\mathcal{O}(2\phi dML_bK)$.

For computational complexity analysis, all three algorithms spend most of the computational resources on GPR. Exact GP inference is quite computationally demanding, requiring $\mathcal{O}(M^3)$ time complexity and $\mathcal{O}(M^2)$ space complexity [49], which prevents us from using more particles. This is also the bottleneck of GP-enhanced resampling. Finally, since GPR is performed twice in our algorithms, the computation complexity is approximately $\mathcal{O}(M^3 + (M-1)^3)$ for all three algorithms. Note: in computing the computational complexity, we assume hyper-parameters training for GP is excluded and that's also not our focus in this thesis.

3.5.2 Discussion

We will evaluate algorithms from the following three perspectives, memory, computation, and communication overhead. In terms of memory, for all algorithms presented in this Chapter, since we only use about one-tenth the number of particles used in the algorithms presented in Chapter 2 (hundreds compared to thousands), the memory consumption is greatly reduced. In terms of computational complexity, we use the execution time of algorithms as the indicator. The algorithms proposed in this Chapter are superior to GL-DPF, and the running time is much less than GM-DPF (explicit data will be given in Chapter 5). In terms of communication overhead, Algorithm 10 requires two average consensuses which bring a huge challenge to the communication overhead while Algorithm 11 abandons the pursuit of consensus and uses the idea of diffusion strategies, which greatly reduces the communication overhead.

3.5.3 Conclusion

The highlights of this Chapter are listed below.

- The GP-enhanced resampling algorithm [10] is first introduced and then exploited to enable the direct particle exchange based DPFs. The core idea behind it is to utilize the variance information provided by GPR, which allows for a better trade-off between exploration and exploitation. The performance improvement of the GP-enhanced resampling algorithm over the standard one is validated in FC-based networks.
- In Section 3.3.2, we removed the fusion center and make our GP-DPF fully distributed. However, the estimation performance is relatively poor and there is a problem of non-convergence.
- In Section 3.4, we made some effort to improve the estimation performance. We implement the improved GP-DPF under two communication strategies, i.e., consensus and diffusive mode. The estimation performance is satisfying for both

strategies, and the communication resource is greatly saved if diffusive mode is adopted. Numerical results verified the feasibility of direct particle exchange based DPF.

- A comparison of the computational and communication complexity of GP-DPFs proposed in this Chapter has be given from a theoretical point of view. Additionally, their limitations and challenges are also discussed.

Metaheuristic Optimization Based Distributed Particle Filtering

4

As a reminder, our goal throughout the thesis is to design an algorithm that allows each agent to approximate the global posterior probability $p(\boldsymbol{x}_n|\boldsymbol{y}_n)$ at each time instant n, represented by M weighted particles. And at time n, our known information includes the estimate of the global posterior probability at each agent k at the previous time instant, denoted as $\{\boldsymbol{x}_{n-1,k}^m, \omega_{n-1,k}^m\}_{m=1,k=1}^{M,K}$, and the measurement information $\boldsymbol{y}_n$ of the whole network, denoted as $\{\boldsymbol{y}_{n,k}\}_{k=1}^K$. When the nature of the quantities exchanged between agents becomes particle, how to design algorithms to optimize the local particle set(e.g., $\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M$) to be consistent with the measurements of the whole network(i.e., $\boldsymbol{y}_n$) becomes the focus of our research. We find it can be regarded as a particle set optimization problem, that is, to seek the global optimal solution of particles through some optimization algorithms. From the optimization perspective, in the absence of communication, each agent that performs particle filtering based on local measurements is effectively stuck in a local optimum. When the agent communicates with other agents, the information located elsewhere in the network is transmitted in the form of particle sets, and the local particle set is continuously evolved through a designed algorithm so that the global optimal solution can be approached. That is the guiding ideology of this Chapter.

Various optimization algorithms can be utilized to achieve the above goal. Due to the similarities between particle filtering and particle swarm optimization (PSO), we are inspired to use modern metaheuristic algorithms to search for globally optimal particle set. In this Chapter, we first give a brief introduction to modern metaheuristic optimization. Next, in Section 4.2, we adopted the famous genetic algorithm [50] in the optimization process of particle set. In Section 4.3, we tried the newer firefly algorithm [51], and the simulation results show that it makes up for the shortcomings of the genetic algorithm, and it has a very satisfactory performance whenever in low communication resources or high communication resources conditions. Finally, in Section 4.4, we analyze the complexity of the algorithms proposed in this Chapter and present a discussion.

4.1 Metaheuristic Optimization

To solve the growing number of optimization problems, metaheuristics are proposed as approximation methods that can find a good enough solution in a reasonable time. The significant difference between heuristic and metaheuristic is that heuristic is more problem-dependent while metaheuristic is considered to be a general algorithmic structure that can be applied to almost all optimization problems [52]. Two major components of metaheuristic algorithms are exploitation and exploration [53], and achieving

the balance between exploitation and exploration is the key to efficient search process. Some properties that characterize most metaheuristics are listed below.

- Metaheuristics are strategies that guide the search process.
- Metaheuristics are problem-independent.
- Metaheuristics are approximate and non-deterministic.
- With metaheuristics, quality solutions to complicated optimization problems can be found in a reasonable time, but there is no guarantee that the optimal solution can be found.

Some of the best-known algorithms include simulated annealing [54], genetic algorithms [50], ant colony optimization [55], particle swarm optimization(PSO) [56], firefly algorithm [51]. Readers can refer to [53] for a brief introduction.

4.2 A Genetic Algorithm Based Distributed Particle Filter

The genetic algorithm is a metaheuristic inspired by the natural biological evolution process, and it consists of three main operators: selection, crossover, and mutation. In recent years, the application of genetic algorithms(GA) in particle filters has been explored. In [42], an evolutionary particle filter has been proposed by using the idea of the genetic algorithm when designing the proposal distribution in the sampling stage. In [44], the genetic algorithm is used in the resampling stage. The particle impoverishment issue has been mitigated in both algorithms. All previous work focused on applying the genetic algorithms in the particle filter, while we extend the idea to distributed particle filter by treating particles at different agents as different parents and fusing the information through the crossover operator. In our case, we combine the GP-DPF proposed in last Chapter with GA.

4.2.1 Genetic Algorithms

Genetic algorithms are population-based optimization methods based on Charles Darwin's theory of natural selection. Three essential components of genetic algorithms are selection, crossover, and mutation.

The selection operator selects candidate solutions according to the law of "survival of the fittest". The quality of the candidate solutions is evaluated by the fitness value, and it reflects the proximity between the candidate solution and the optimal solution. The selected solutions are then fed into the mating pool to await the crossover operation. Next, the crossover operator randomly selects two candidate solutions (i.e., parents) from the mating pool and exchanges their partial information to create a new solution (i.e., offspring). Like individuals in nature, the offspring in the genetic algorithm may also mutate. The mutation operator is used to change part of the information in the offspring, and this is very important since maintaining the diversity of the population can prevent the genetic algorithm from falling into the local optimal solution.

The genetic algorithm used in the DPF in this Section refers to [44].

4.2.2 GA-DPF under Gossip Protocol

In this Section, we consider the use of a classical genetic algorithm, where each offspring has exactly two parents. Therefore, we choose the gossip communication protocol in this Section, that is, at each moment, each agent can only communicate with one agent connected to it.

The proposed algorithm contains five stages, i.e., local filtering, communication, mating, crossover, and mutation(the flowchart is given in Fig 4.1). Each stage is discussed in the following. Note, that the selection rule is omitted here since we assume particles with high "fitness" have already been screened out after applying GP-enhanced resampling.

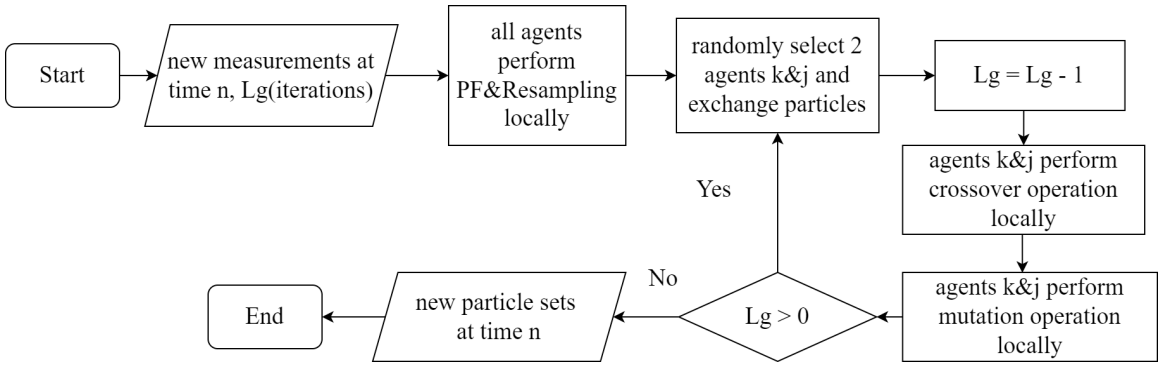


Figure 4.1: The flowchart of the GA-DPF under gossip protocol.

Local filtering

In this stage, each agent performs the particle filtering as well as the GP-enhanced resampling based on the newly acquired measurements and ends up with a set of particles $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$.

Communication

In this stage, the gossip communication protocol is adopted, and particle exchange is executed. Taking agent k as an example, let agent j be one of its neighbors and be selected, then the agent k would hold a particle set denoted as $\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M$ after the communication stage.

Mating

To make full use of the two particle sets, we first perform a mating operation before crossover. Again take agent k as an example, we first randomly select a particle from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ as one of the parents, and then randomly select a particle from $\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$ as the other parent. In this way, we can ensure that two parents come from different particle sets, thus achieving the goal of information fusion. Note that after we select a particle from a particle set, we need to remove that particle from the particle set to

which it belongs. This ensures that all particles can be selected once. So far, we have M pairs of successfully mated parents.

Crossover

Crossover is performed to fuse the information from different agents and prevent the algorithm from trapping into the local optimum. For ease of explanation, at agent k, we take a pair from the M pairs of parents, denoted as $\{\boldsymbol{x}^{\text{par},1}, \boldsymbol{x}^{\text{par},2}\}$. Note, the fitness of a particle is defined as its local measurement likelihood. For example, the fitness of particle $\boldsymbol{x}^{\text{par},1}$ is defined in (4.1). The superscript of f is consistent with the superscript of its corresponding particle.

$$f^{\text{par},1} = p(\boldsymbol{y}_k|\boldsymbol{x}^{\text{par},1}), \tag{4.1}$$

Then, we apply the arithmetic crossove with a probability p_c. After the crossover operation, an offspring particle is generated and can be written as

$$\boldsymbol{x}^{\text{coff}} = \alpha\boldsymbol{x}^{\text{par},1} + (1-\alpha)\boldsymbol{x}^{\text{par},2} \tag{4.2}$$

where $\alpha = f^{\text{par},1}/(f^{\text{par},1} + f^{\text{par},2})$. The fitness of the offspring should also be calculated and is denoted as f^{coff}. The crossover probability p_c is determined adaptively using the Sigmoid function and is given by [44]

$$p_c = \begin{cases} p_{c1} & f' < f_{\text{avg}} \\ p_{c2} - \frac{p_{c2}-p_{c1}}{1+\exp\{\lambda[\frac{2(f'-f_{\text{avg}})}{f_{\max}-f_{\text{avg}}}-1]\}} & f \geq f_{\text{avg}} \end{cases}, \tag{4.3}$$

where p_{c1}, p_{c2} are the predefined upper and lower bounds of crossover probability. λ is a determined parameter. $f_{\max}, f_{\text{avg}}$ are the maximum and average fitness values of the parental particles. f' is the larger fitness value of two selected parents, i.e., $f' = \max\{f^{\text{par},1}, f^{\text{par},2}\}$. The benefit of using this adaptively changing crossover probability is that when particles run the risk of prematurely converging to a local optimum (i.e., $f' \approx f_{\text{avg}}$), p_c increases; when the particles are at risk of divergence in the solution space(i.e., $f' \approx f_{\max}$), p_c decreases. By doing so, the algorithm is more robust and would have a higher probability for finding the global optimum. And the offspring is accepted based on the Metropolis rule [57]. The idea of the rule is that the degraded offspring (i.e., with poorer fitness value) is accepted with a certain probability. The specific operation process is as follows. If $f^{\text{coff}} > f'$, the offspring is accepted. Otherwise, it is accepted with the probability f^{coff}/f'. It is realized by generating a random number ζ from a standard uniform distribution and comparing it with f^{coff}/f'. If $\zeta < f^{\text{coff}}/f'$, the offspring is accepted, otherwise, it is rejected. The new generated particle set should include the offspring $\boldsymbol{x}^{\text{coff}}$(if accepted) or the parent $\boldsymbol{x}^{\text{par},1}$ sampling from the original particle set $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$(if rejected). The pseudo-code for standard crossover is presented in Algorithm 14.

Mutation

The mutation operation aims at giving some randomness to the particles to enable them to get rid of the local optimum. For each particle in the newly generated particle

set after the crossover operation, the mutation is performed with the probability p_m, given by

$$\boldsymbol{x}_k^{m,c} = \boldsymbol{x}_k^m + \boldsymbol{\varepsilon}, \tag{4.4}$$

where $\boldsymbol{x}_k^m$ is the m^{th} particle from the newly generated particle set while $\boldsymbol{x}_k^{m,c}$ is the particle after applying the mutation. $\boldsymbol{\varepsilon}$ is the noise generated from from the same distribution as the process noise. The mutation probability p_m is given by

$$p_m = \begin{cases} p_{m1} & f < f_{\text{avg}} \\ p_{m2} - \frac{p_{m2}-p_{m1}}{1+\exp\{\lambda[\frac{2(f-f_{\text{avg}})}{f_{\max}-f_{\text{avg}}}-1]\}} & f \geq f_{\text{avg}} \end{cases}, \tag{4.5}$$

where p_{m1}, p_{m2} are the predefined upper and lower bounds of mutation probability. λ is a determined parameter. $f_{\max}, f_{\text{avg}}$ are the maximum and average fitness values of particles after crossover operation. f is the fitness value of $\boldsymbol{x}_k^m$. The fitness value for $\boldsymbol{x}_k^{m,c}$ should be also calculated and is denoted as f^c(the calculation procedure can refer to (4.1)).The mutated particle is accepted based on the Metropolis rule. If $f^c > f$, the particle is accepted. Otherwise, it is accepted with the probability f^c/f. The accepted mutated particle $\boldsymbol{x}_k^{m,c}$ should replace $\boldsymbol{x}_k^m$ in the new generated particle set. The pseudo-code for mutation is presented in Algorithm 15. So far, after the above five stages, one genetic algorithm execution is completed. The particle set of agent k has completed an evolution by absorbing the information from agent j.

The algorithm GA-DPF under gossip protocol is presented in Algorithm 12. Stage mating, crossover and mutation form the genetic algorithm based fusion(GAF) and is presented in Algorithm 13. The parameter settings are given in Table 4.1.

Algorithm 12 GA-DPF(gossip protocol)

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K$

Initialize M, L_g(iterations)

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}) \qquad \forall k = 1, \cdots, K$

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{GP\text{-}R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M) \qquad \forall k = 1, \cdots, K$

for $i = 1, \cdots, L_g$ **do**

    Randomly select 2 adjacent sensors k, j

    Sensor k, j exchange their particle sets

    Run parallel at sensors k, j:

$$\begin{cases} \{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \mathbf{GAF}(\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}) \\ \{\boldsymbol{x}_{n,j}^m\}_{m=1}^M = \mathbf{GAF}(\{\boldsymbol{x}_{n,j}^m, \boldsymbol{x}_{n,k}^m\}_{m=1}^M, \boldsymbol{y}_{n,j}) \end{cases}$$

end for

$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M) \qquad \forall k = 1, \cdots, K$

return $\{\boldsymbol{x}_{n,k}\}_{k=1}^K, \{\boldsymbol{x}_{n,k}^m\}_{m=1,k=1}^{M,K}$

Algorithm 13 Genetic algorithm based fusion(GAF)

Require: $\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}$

Calculate particle weight $\{f_k^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,k}^m), f_j^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,j}^m)\}_{m=1}^M$

$\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \textbf{SCrossover}(\{\boldsymbol{x}_{n,k}^m, f_k^m, \boldsymbol{x}_{n,j}^m, f_j^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$

Calculate particle weight $\{f^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,k}^m)\}_{m=1}^M$

$\{\hat{\boldsymbol{x}}_{n,k}^m\}_{m=1}^M = \textbf{Mutation}(\{\boldsymbol{x}_{n,k}^m, f^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$

return $\{\hat{\boldsymbol{x}}_{n,k}^m\}_{m=1}^M$

Algorithm 14 Standard Crossover(SCrossover)

Require: $\{\boldsymbol{x}_{n,k}^m, f_k^m, \boldsymbol{x}_{n,j}^m, f_j^m\}_{m=1}^M, \boldsymbol{y}_{n,k}$

Initialize p_{c1}, p_{c2}, λ

Calculate $f_{\max} = \max(\{f_k^m, f_j^m\}_{m=1}^M), f_{\text{avg}} = mean(\{f_k^m, f_j^m\}_{m=1}^M)$

for $s = 1, \cdots, M$ **do**

    Generate a pair $\{\boldsymbol{x}^{\text{par},1}, \boldsymbol{x}^{\text{par},2}\}$ by randomly selecting two particles from $\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M$

    Calculate $f' = \max(f^{\text{par},1}, f^{\text{par},2})$

    Calculate crossover probability p_c according to (4.3)

    Calculate $\alpha = f^{\text{par},1}/(f^{\text{par},1} + f^{\text{par},2})$

    if $\mathcal{U}(0,1) < p_c$ **then**

        $\boldsymbol{x}^{\text{coff}} = \alpha \boldsymbol{x}^{\text{par},1} + (1-\alpha)\boldsymbol{x}^{\text{par},2}$

        Calculate f^{coff}

        if $f^{\text{coff}} > f'$ **then** ▷ accepted based on Metropolis rule

            Replace $\boldsymbol{x}^{\text{par},1}$ in $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ with $\boldsymbol{x}^{\text{coff}}$

        else

            if $\mathcal{U}(0,1) < f^{\text{coff}}/f'$ **then**

                Replace $\boldsymbol{x}^{\text{par},1}$ in $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ with $\boldsymbol{x}^{\text{coff}}$

            end if

        end if

    end if

end for

return $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

Algorithm 15 Mutation

Require: $\{\boldsymbol{x}_{n,k}^m, f^m\}_{m=1}^M, \boldsymbol{y}_{n,k}$
Initialize $p_{m1}, p_{m2}, \lambda, \Sigma$
Calculate $f_{\max} = \max(\{f^m\}_{m=1}^M), f_{\text{avg}} = mean(\{f^m\}_{m=1}^M)$
for $s = 1, \cdots, M$ **do**
    Select a particle $\boldsymbol{x}_{n,k}^s$ from $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ and determine its weight f
    Calculate mutation probability p_m according to (4.5)
    if $\mathcal{U}(0,1) < p_m$ **then**
        $\boldsymbol{x}^c = \boldsymbol{x}_{n,k}^s + \varepsilon$, $\varepsilon \sim \mathcal{N}(\mathbf{0}, \Sigma)$
        Calculate weight f^c corresponding to $\boldsymbol{x}^c$
        if $f^c > f$ **then** ▷ accepted based on Metropolis rule
            $\hat{\boldsymbol{x}}_{n,k}^s = \boldsymbol{x}^c$
        else
            if $\mathcal{U}(0,1) < f^c/f$ **then**
                $\hat{\boldsymbol{x}}_{n,k}^s = \boldsymbol{x}^c$
            else
                $\hat{\boldsymbol{x}}_{n,k}^s = \boldsymbol{x}_{n,k}^s$
            end if
        end if
    end if
end for
return $\{\hat{\boldsymbol{x}}_{n,k}^m\}_{m=1}^M$

Table 4.1: Simulation setup of GA-DPF.

parameter	value
λ	9.903438
p_{c1}	0.9
p_{c2}	0.6
p_{m1}	0.1
p_{m2}	0.01
Σ	$\boldsymbol{R}$(see (1.15))
Monte Carlo trials	50x

The simulation results of GA-DPF given in Fig 4.2 show the effect of the number of gossip iterations on the tracking error under three different particle set sizes. From the figure we can see that the larger the number of particles, the smaller the error. At the same time, as the number of gossip iterations increases, the tracking error gradually decreases. We can surprisingly find that even when we set the local particle size of each agent to a very small size, such as 10, the error can reduce to very small as long as there are enough iterations. However, too many iterations will make the algorithm difficult to apply to scenarios with high real-time requirements.

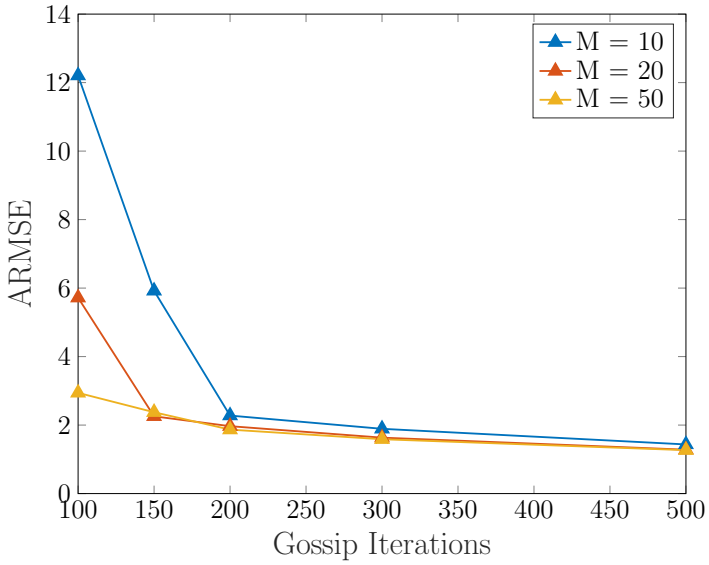


Figure 4.2: Algorithm:GA-DPF(gossip-based)-ARMSE as a function of local particle size and the number of iterations.

4.2.3 GA-DPF under Local Broadcast Protocol

In the previous Section, the proposed algorithm can only adapt to the gossip communication protocol, which means only information from two agents can be fused at a time. At this Section, we make a few improvements to make the proposed GA-DPF adapt to a more general communication protocol by exploiting multi-parent crossover operators. Hence, local broadcast communication is used in this Section.

The modified algorithm also contains the same five stages but is slightly different except for local filtering and mutation. Here we only briefly introduce the left three stages. The flowchart of the algorithm is depicted in Fig 4.3.

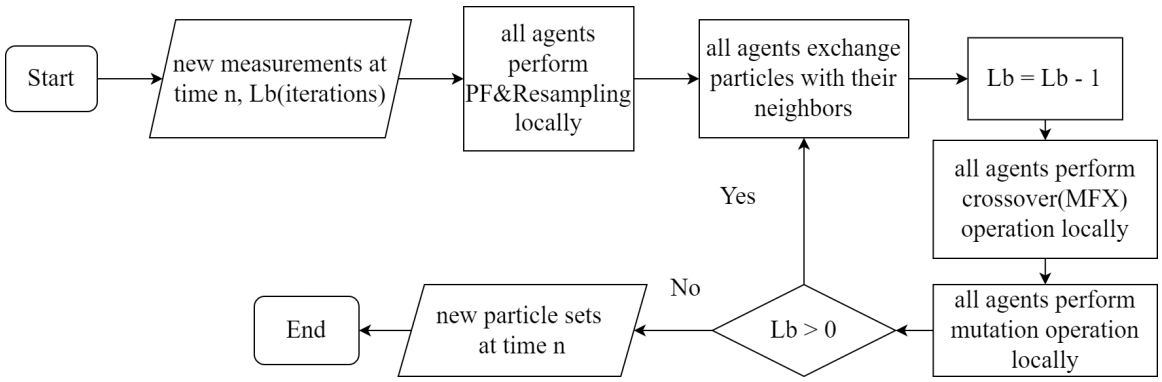


Figure 4.3: The flowchart of the GA-DPF under local broadcast protocol.

Communication

Since now an agent can communicate with several connected agents at a time, an offspring would simultaneously have multiple parents. Take agent k as an example and let $\mathcal{N}_k$ denotes the neighbors of agent k. After one communication iteration, the agent would hold a particle set denoted as $\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$.

Mating

This is the same as Section 4.2.2, except that the M pairs of parents consist of multiple particles each.

Crossover

In recent years, several multi-parent crossover operators for real-coded Genetic algorithms have been proposed [58][59], such as seed crossover operator(SX), multi-parent feature-wise crossover operator(MFX), center of mass crossover operator(CMX) etc. We test some of them in our application and pick the best performing one to show here, which is MFX. First we take a pair of parents from agent k, denoted as $\{\boldsymbol{x}^{\text{par},1},\cdots,\boldsymbol{x}^{\text{par},D_k+1}\}$, where D_k is the number of agent k's neighbors. Then, we choose an individual $\boldsymbol{x}^{\text{par},v}(v=1,\cdots,D_k+1)$ from the set and one offspring is generated based on the following procedure. For each $w(w=1,\cdots,d)$, where d is the dimension of the state space, again we choose a particle $\boldsymbol{x}^{\text{par},u}$, where u is randomly drawn from $\{1,\cdots,D_k+1\}$ except for v. Then one dimension of the offspring is calculated as follows,

$$x_w^{\text{off},v}=\alpha x_w^{\text{par},v}+(1-\alpha)x_w^{\text{par},u}, \tag{4.6}$$

where $\alpha=f^{\text{par},v}/(f^{\text{par},v}+f^{\text{par},u})$. The behind idea is to employ multiple parents for generating one offspring using different feature-wise information and the (4.6) has to be executed d times to generate an offspring. For the new generated offspring $\boldsymbol{x}^{\text{off},v}$, the corresponding fitness value is denoted as f^v(the calculation procedure can refer to (4.1)). As we keep one parent fixed for generating a new offspring, we end up with a total of D_k+1 offsprings for each pair of parents. Since only one offspring is required for each pair of parents, we then keep the offspring with the highest fitness value, i.e.,

$$v^*=\arg\min_v f^v \tag{4.7}$$

$$\boldsymbol{x}^{\text{coff}}=\boldsymbol{x}^{\text{off},v^*} \tag{4.8}$$

The definition of the probability to perform a crossover is the same as that in the previous Section. Besides, the newly generated offspring is also accepted according to the Metropolis rule. The pseudo-code for multi-parent crossover is presented in Algorithm 16.

The algorithm GA-DPF under local broadcast protocol is presented in Algorithm 18. Stage mating, multi-parent crossover and mutation form the multi-parent genetic algorithm based fusion(MGAF) and is presented in Algorithm 17.

Algorithm 16 Multi-parent Crossover(MCrossover)

Require: $\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m, f_j^m\}_{m=1}^M, \boldsymbol{y}_{n,k}$

Initialize $p_{c1}, p_{c2}, \lambda, d, D_k$
Calculate $f_{\max} = \max(\{f_j^m\}_{j,m=1}^{\{k,\mathcal{N}_k\},M})$, $f_{\text{avg}} = mean(\{f_j^m\}_{j,m=1}^{\{k,\mathcal{N}_k\},M})$
for $s = 1, \cdots, M$ **do**
    Generate parents pool $\{\boldsymbol{x}_{\text{pool}}^u\}_{u=1}^{D_k+1}$ by selecting $D_k + 1$ particles from $\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$
    Calculate $f' = \max(\{f^{\text{par},u}\}_{u=1}^{D_k+1})$
    Calculate crossover probability p_c according to (4.3)
    if $\mathcal{U}(0,1) < p_c$ **then**
        for $v = 1, \cdots, D_i$ **do**
            $\boldsymbol{x}^{\text{par},1} = \boldsymbol{x}_{\text{pool}}^v$
            for $w = 1, \cdots, d$ **do**
                determine $\boldsymbol{x}^{\text{par},2}$ by randomly choosing a particle from $\{\boldsymbol{x}_{\text{pool}}^u\}_{u=1}^{\{D_k+1\setminus v\}}$
                $x_w^{\text{off},v} = f^{\text{par},1}/(f^{\text{par},1} + f^{\text{par},2}) * x_w^{\text{par},1} + f^{\text{par},2}/(f^{\text{par},1} + f^{\text{par},2}) * x_w^{\text{par},2}$
            end for
            Compute the $f^{\text{off},v}$ corresponding to the new offspring $\boldsymbol{x}^{\text{off},v}$
        end for
        Select the particle with $\max\{f^{\text{off},v}\}_{v=1}^{D_k+1}$ as the final offspring $\boldsymbol{x}^{\text{coff}}$
    end if
    Calculate f^{coff}
    if $f^{\text{coff}} > f'$ **then** ▷ accepted based on Metropolis rule
        Accept the generated offspring $\boldsymbol{x}^{\text{coff}}$ and replace the original particle in $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$
    else
        if $\mathcal{U}(0,1) < f^{\text{coff}}/f'$ **then**
            Accept the generated offspring $\boldsymbol{x}^{\text{coff}}$ and replace the original particle in $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$
        end if
    end if
end for
return $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

Algorithm 17 Multi-parent Genetic algorithm based fusion(MGAF)

Require: $\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}$

Calculate particle weight $\{f_j^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,j}^m)\}_{j,m=1}^{\{k,\mathcal{N}_k\},M}$
$\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \textbf{MCrossover}(\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m, f_j^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$
Calculate particle weight $\{f^m = p(\boldsymbol{y}_{n,k}|\boldsymbol{x}_{n,k}^m)\}_{m=1}^M$
$\{\hat{\boldsymbol{x}}_{n,k}^m\}_{m=1}^M = \textbf{Mutation}(\{\boldsymbol{x}_{n,k}^m, f^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$
return $\{\hat{\boldsymbol{x}}_{n,k}^m\}_{m=1}^M$

Algorithm 18 GA-DPF(local broadcast protocol)

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K$

Execute at all agents $k = 1, \cdots, K$ in parallel:

Initialize M, L_b(iterations)

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \textbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \textbf{GP-R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M)$

Sensor k sends $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ to its neighbors

for $i = 1, \cdots, L_b$ **do**

  $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \textbf{MGAF}(\cup_{j\in\{k,\mathcal{N}_k\}}\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M, \boldsymbol{y}_{n,k})$

end for

$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M)$

return $\{\boldsymbol{x}_{n,k}\}_{k=1}^K, \{\boldsymbol{x}_{n,k}^m\}_{m=1,k=1}^{M,K}$

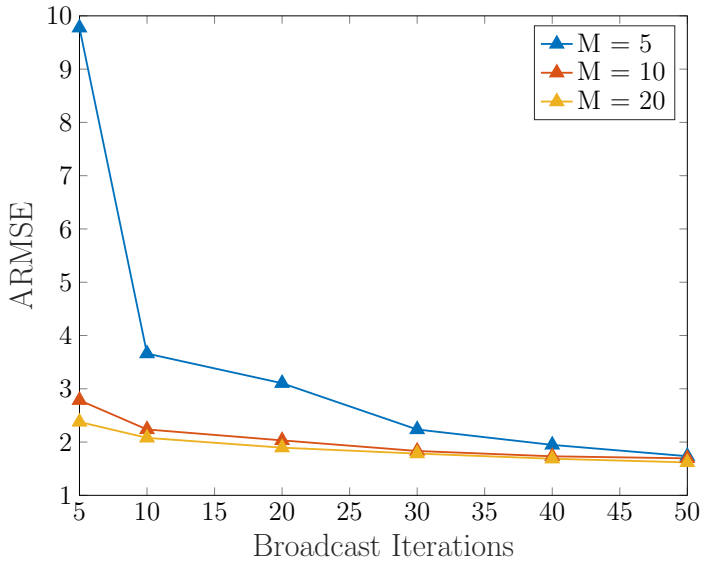


Figure 4.4: Algorithm:GA-DPF(local broadcast-based)-ARMSE as a function of local particle size and the number of iterations.

The simulation results of GA-DPF under local broadcast communication protocol given in Fig 4.4 show the effect of the number of broadcast iterations on tracking error for three different particle set sizes. Increasing the particle set or increasing the number of broadcast iterations can both reduce the tracking error. At the same time, the algorithm does not have high requirements on the size of the particle set and the number of iterations. As can be seen from the figure, 10 local particles and 10 iterations can already reduce the ARMSE to around 2. Different from GA-DPF based on the gossip communication protocol in the previous Section, since all the agents can perform information fusion with their connected agents once in each iteration under this structure, ARMSE can be significantly reduced after the first few iterations. Besides, execution time can be reduced due to the fact that all agents communicate in parallel

in each iteration. It is worth mentioning that in the simulation, only after all agents in the whole network have completed the communication and computing tasks, can they together enter the next iteration. Hence, the performance of the algorithm under the condition of "asynchronous" communication (for example, when an agent observes that all the connected agents have completed the current computing task, it directly go to the next iteration without waiting for the rest of the agents in the network) can be further investigated.

4.3 A Firefly Algorithm Based Distributed Particle Filter

In recent years, inspired by the behaviors of the flashing characteristics of fireflies, a new bionic algorithm, Firefly Algorithm(FA), is proposed by Yang [51], and its superiority to GA and PSO algorithms is confirmed by simulation experiments. In [45], FA is also applied to the particle filter to prevent particle impoverishment and achieves satisfactory results with a limited number of particles. Since FA outperforms GA in other optimization applications, in this Section, we are motivated to adopt the FA in DPF by treating particles as fireflies. In our case, we combine the GP-DPF proposed in last Chapter with FA.

4.3.1 Firefly Algorithm

The Firefly Algorithm (FA) mimics the social behavior of fireflies flying through the sky and is based on the following idealized behavior of firefly flashing characteristics:

- All fireflies are unisex, so one firefly will be attracted to other fireflies regardless of their sex.
- Attractiveness is proportional to their brightness and they both decrease with distance, so for any two flashing fireflies, the less bright one will move towards the brighter one. If there is no brighter firefly than a particular firefly, it will move randomly.
- The brightness of fireflies is influenced or determined by the landscape of the objective function.

Attractiveness

The brightness I of a firefly at a certain location $\boldsymbol{x}$ is determined by the objective function, i.e., $I(\boldsymbol{x}) \propto f(\boldsymbol{x})$. However, the attractiveness β is relative and it should be judged by other fireflies. For example, if we have 2 fireflies k and j, then the attractiveness of the brighter of the two to the other should vary with the distance r_{kj} between the two, and the relationship usually can be described by the following formula

$$\beta(r_{kj}) = \beta_0 e^{-\gamma r_{kj}^2}. \tag{4.9}$$

Since we can approximate exponential function as $1/(1 + r^2)$ to save computation resources, the above equation can be rewritten as

$$\beta(r_{kj}) = \beta_0/(1 + \gamma r_{kj}^2). \tag{4.10}$$

where β_0 is the attractiveness when $r_{kj} = 0$ and γ is called light absorption coefficient, which determines the strength of the influence of distance on attractiveness. The distance can be defined using the Cartesian distance,

$$r_{kj} = ||\boldsymbol{x}_k - \boldsymbol{x}_j||_2 = \sqrt{\sum_{i=1}^{d}(x_k^i - x_j^i)^2}, \tag{4.11}$$

where x_k^i is the i^{th} component of vector $\boldsymbol{x}_k$ of the k^{th} firefly and d is the dimension.

Movement

The movement of a firefly k towards a brighter firefly j can be modeled as

$$\boldsymbol{x}_k = \boldsymbol{x}_k + \beta_0 e^{-\gamma r_{kj}^2}(\boldsymbol{x}_j - \boldsymbol{x}_k) + \alpha \boldsymbol{S}(\text{rand} - \frac{1}{2}), \tag{4.12}$$

where the second term is due to the attractiveness and the third term is a random term. $\boldsymbol{S} \in \mathbb{R}^{d\times d}$ is the scaling matrix whose dimension is consistent with that of term rand. In the original paper, rand follows a uniform distribution between 0 and 1. For most cases, we can take $\beta_0 = 1$ and $\alpha \in [0, 1]$. Furthermore, the random term rand can also be extended to other distributions like Gaussian. Without the random term, the system will evolve in a deterministic way. Conversely, the use of random term gives FA the ability to explore the search space, helping agent get rid of local optima. Therefore, it is possible to achieve a good balance between local intensive exploitation and global exploration by adjusting α.

4.3.2 FA-DPF under Gossip Protocol

Here we explain how we apply FA to direct particle exchange based distributed particle filtering. The algorithm FA-DPF is presented in Algorithm 19.

In this Section, the gossip communication protocol is adopted. For convenience, in all the descriptions that follow, we default that in one iteration, agent k and agent j are selected. Before we enter the communication stage, all agents need to perform the particle filtering as well as GP-enhanced resampling locally same as the previous Sections. After an iteration, the two selected agents will both have each other's particle sets and the final set is $\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M$. Since we want to fuse the information from agent k and j, what we need to perform locally is, taking agent k as an example, letting the particles in set $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ move towards to the "brighter" particles in set $\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$. Now the remaining question is how to define the brightness of a certain particle.

The deviation of the particle from the true state can be measured by the local measurement likelihood. Based on this, we wish to define the brightness of a particle as a function of the likelihood value. However, if only particle sets are exchanged in the communication process, only local measurements can be used to calculate the likelihood of locally generated or received particle sets. In this Section, to make the defined brightness more convincing, in addition to exchanging particles, each agent also needs to exchange the likelihood values of all particles calculated based on its local

measurements. To sum up, each agent needs to perform two communication rounds in each iteration, and each iteration consists of the following four stages. Note, we only describe the behavior of agent k, the same for agent j. The flowchart of the algorithm is depicted in Fig 4.5.

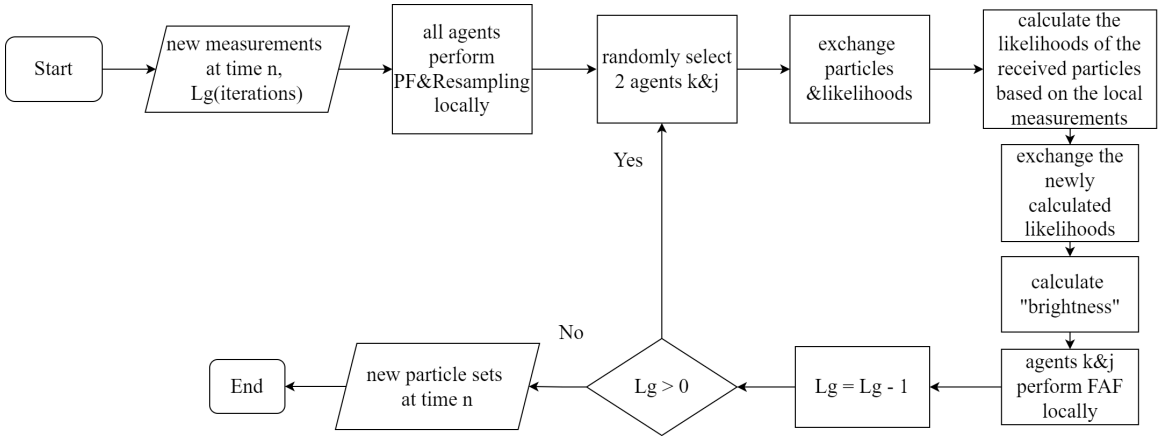


Figure 4.5: The flowchart of the FA-DPF under gossip protocol.

Communication round 1

Before the communication round, agent k holds data $\{\boldsymbol{x}_{n,k}^m, \omega_{n,kk}^m\}_{m=1}^M$, where the first k of the subscript of ω means the particles are from agent k while the second k means the likelihood is calculated based on the measurements from agent k. In the first communication round, agent k would send the data $\{\boldsymbol{x}_{n,k}^m, \omega_{n,kk}^m\}_{m=1}^M$ to agent j and receive $\{\boldsymbol{x}_{n,j}^m, \omega_{n,jj}^m\}_{m=1}^M$ from agent j. Before the next communication round, agent k needs to calculate the likelihood of the received particles $\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$ based on its local measurements and end up with a data set $\{\omega_{n,jk}^m\}_{m=1}^M$.

Communication round 2

In this communication round, agent k should send $\{\omega_{n,jk}^m\}_{m=1}^M$ to agent j and receive $\{\omega_{n,kj}^m\}_{m=1}^M$ from agent j. Till now, the communication procedure has been finished. The data hold by agent k is $\{\boldsymbol{x}_{n,k}^m, \omega_{n,kk}^m, \omega_{n,kj}^m, \boldsymbol{x}_{n,j}^m, \omega_{n,jj}^m, \omega_{n,jk}^m\}_{m=1}^M$.

Brightness calculation

Here we give the definition of brightness. For particle $\boldsymbol{x}_{n,k}^m$, the brightness $I(\boldsymbol{x}_{n,k}^m) = \omega_{kk}^m * \omega_{kj}^m$.

Firefly algorithm based fusion

After determining the particles $\{\boldsymbol{x}_{n,k}^m, \boldsymbol{x}_{n,j}^m\}_{m=1}^M$ and the corresponding brightness $\{I(\boldsymbol{x}_{n,k}^m), I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M$, we can now introduce the FA. First, we draw a particle $\boldsymbol{x}_{n,k}^p$ from

the set $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ and make a comparison between $I(\boldsymbol{x}_{n,k}^p)$ with $\max(\{I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M)$. If $I(\boldsymbol{x}_{n,k}^p) < \max(\{I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M)$, we randomly draw a particle $\boldsymbol{x}_{n,j}^q$ from the set $\{\boldsymbol{x}_{n,j}^m\}_{m=1}^M$ meeting the requirement that $I(\boldsymbol{x}_{n,j}^q) > I(\boldsymbol{x}_{n,k}^p)$. The above operations can be performed efficiently by introducing a sorting algorithm. Then we move the particle $\boldsymbol{x}_{n,k}^p$ towards $\boldsymbol{x}_{n,j}^q$ based on the following movement rule

$$\boldsymbol{x}_{n,k}^p = \boldsymbol{x}_{n,k}^p + \beta_0/(1+\gamma r_{pq}^2)(\boldsymbol{x}_{n,j}^q - \boldsymbol{x}_{n,k}^p) + \alpha\varepsilon, \tag{4.13}$$

where

$$r_{pq} = ||\boldsymbol{x}_{n,k}^p - \boldsymbol{x}_{n,j}^q||_2. \tag{4.14}$$

As you can see (4.13) is slightly different with (4.12), it is because we use the process noise $\varepsilon \sim \mathcal{N}(0, \boldsymbol{R})$ to replace the random item $\boldsymbol{S}(\text{rand} - \frac{1}{2})$. The above operation needs to be repeated M times until each particle in the particle set $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$ has been processed. So far, one iteration is finished. We can perform multiple iterations to ensure information from the entire network is fully fused. The pseudo-code for firefly algorithm based fusion is presented in Algorithm 20.

Algorithm 19 FA-DPF

Require: $\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \{\boldsymbol{y}_{n,k}\}_{k=1}^K$

Initialize M, L_g(iterations)

$\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M = \mathbf{PF}(\{\boldsymbol{x}_{n-1}^m, \omega_{n-1}^m\}_{m=1}^M, \boldsymbol{y}_{n,k}) \qquad \forall k = 1, \cdots, K$

$\{\boldsymbol{x}_{n,k}^{m)}\}_{m=1}^M = \mathbf{GP\text{-}R}(\{\boldsymbol{x}_{n,k}^m, \omega_{n,k}^m\}_{m=1}^M) \qquad \forall k = 1, \cdots, K$

Calculate the particles' weight $\{\omega_{n,kk}^m\}_{m=1}^M \qquad \forall k = 1, \cdots, K$

for $i = 1, \cdots, L_g$ **do**

  Randomly select 2 adjacent sensors k, j

  Sensor k, j exchange their particles and weights $\{\omega_{n,kk}^m\}_{m=1}^M, \{\omega_{n,jj}^m\}_{m=1}^M$

  Run parallel at sensors k, j:

$$\begin{cases} \text{Calculate weight } \omega_{n,kj}^m \text{ of set } \{\boldsymbol{x}_{n,k}^m\}_{m=1}^M \\ \text{Calculate weight } \omega_{n,jk}^m \text{ of set } \{\boldsymbol{x}_{n,j}^m\}_{m=1}^M \end{cases}$$

  Sensor k, j exchange weights $\{\omega_{n,jk}^m\}_{m=1}^M, \{\omega_{n,kj}^m\}_{m=1}^M$

  Run parallel at sensors k, j:

$$\begin{cases} \{I(\boldsymbol{x}_{n,k}^m)\}_{m=1}^M = \{\omega_{n,kk}^m * \omega_{n,kj}^m\}_{m=1}^M \\ \{I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M = \{\omega_{n,jk}^m * \omega_{n,jj}^m\}_{m=1}^M \end{cases}$$

  Run parallel at sensors k, j:

$$\begin{cases} \{\boldsymbol{x}_{n,k}^m\}_{m=1}^M = \mathbf{FAF}(\{\boldsymbol{x}_{n,k}^m, I(\boldsymbol{x}_{n,k}^m)\}_{m=1}^M, \{\boldsymbol{x}_{n,j}^m, I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M) \\ \{\boldsymbol{x}_{n,j}^m\}_{m=1}^M = \mathbf{FAF}(\{\boldsymbol{x}_{n,j}^m, I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M, \{\boldsymbol{x}_{n,k}^m, I(\boldsymbol{x}_{n,k}^m)\}_{m=1}^M) \end{cases}$$

end for

$\boldsymbol{x}_{n,k} = mean(\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M) \qquad \forall k = 1, \cdots, K$

return $\{\boldsymbol{x}_{n,k}\}_{k=1}^K, \{\boldsymbol{x}_{n,k}^m\}_{m=1,k=1}^{M,K}$

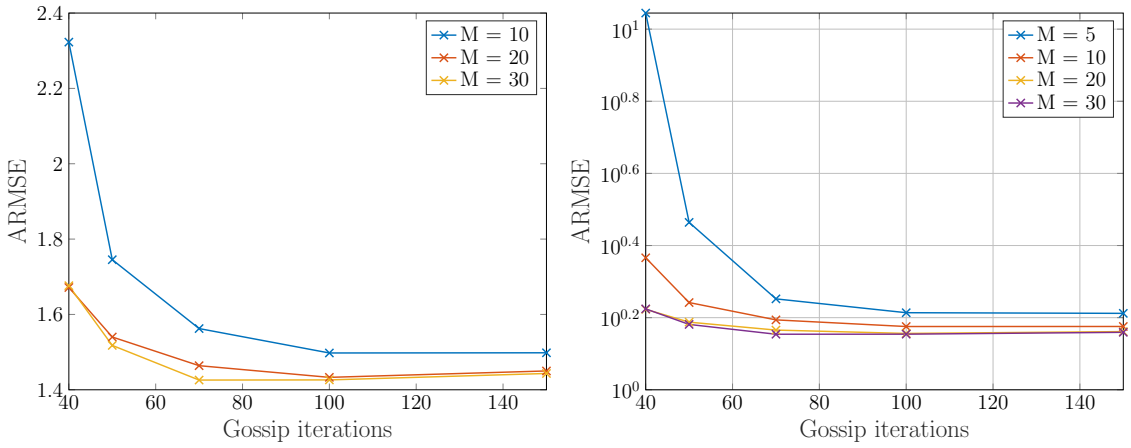
Algorithm 20 Firefly algorithm based fusion(FAF)

Require: $\{\boldsymbol{x}_{n,k}^m, I(\boldsymbol{x}_{n,k}^m)\}_{m=1}^M, \{\boldsymbol{x}_{n,j}^m, I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M$

Initialize β_0, α, γ
for $p = 1, \cdots, M$ **do**
  if $\max(\{I(\boldsymbol{x}_{n,j}^m)\}_{m=1}^M) > I(\boldsymbol{x}_{n,k}^p)$ **then**
    Randomly draw a particle $\boldsymbol{x}_{n,j}^q$ satisfying $I(\boldsymbol{x}_{n,j}^q) > I(\boldsymbol{x}_{n,k}^p)$
    $r_{pq} = ||\boldsymbol{x}_{n,k}^p - \boldsymbol{x}_{n,j}^q||_2$
    $\boldsymbol{x}_{n,k}^p = \boldsymbol{x}_{n,k}^p + \beta_0/(1+\gamma r_{pq}^2)(\boldsymbol{x}_{n,j}^q - \boldsymbol{x}_{n,k}^p) + \alpha\varepsilon$
  end if
end for
return $\{\boldsymbol{x}_{n,k}^m\}_{m=1}^M$

Table 4.2: Simulation setup of FA-DPF.

parameter	value
α	1
γ	0.03
β_0	0.5
Monte Carlo trials	100x



(a) y-axis in linear scale.

(b) y-axis in log scale.

Figure 4.6: Algorithm:FA-DPF(gossip-based)-ARMSE as a function of local particle size and the number of iterations.

The parameter settings are given in Table 4.2, and Fig 4.6 shows the simulation results of FA-DPF. From the figure we can see that the ARMSE can be reduced by increasing the particle set size or increasing the number of iterations. Compared with

GA-DPF, the number of iterations required is greatly reduced. However, since FA-DPF requires two communication rounds (particles and likelihoods) in sequence during one iteration, this imposes higher requirements on the design of the communication protocol. In addition, if you look closely at the figures, you will find that when the number of iterations continues to increase after a certain number of times, the ARMSE tends to saturate and rise slightly, this is because the measurement functions only depend on the first 4 dimensions of the state space in our simulation data model. Therefore, in each iteration, the estimations on the other two dimensions cannot be corrected, leading to the increasing variance due to the presence of the random term, resulting in a slight increase in the overall error. If we only calculate the error of the first four dimensions from the true state, then the ARMSE always decreases as the number of iterations increases. In the future, how to define brightness can be further explored.

4.4 Summary

In this Section, we will briefly analyze and discuss the DPF algorithms proposed in this Chapter.

4.4.1 Complexity analysis

Table 4.3: Communication overhead and computational complexity analysis(Chapter 4)
(M: the size of the particle set; d: the dimension of the state space; $D_{\max}$: the maximum degree of the given multi-agent network; L_b: the number of broadcast iterations; K: the number of agents; L_g: the number of gossip iterations.)

Algorithm	GA-DPF(gossip)	GA-DPF(broadcast)	FA-DPF(gossip)
communication	$\mathcal{O}(2L_gMd/K)$	$\mathcal{O}(2L_bMd) - \mathcal{O}(2KL_bMd)$	$\mathcal{O}(L_gM(2d+4)/K)$
computation	$\mathcal{O}(2L_gMd/K)$	$\mathcal{O}(L_bMdD_{\max})$	$\mathcal{O}(2L_gMlog(M)d/K)$

- GA-DPF(gossip)
  First we focus on the calculation on communication complexity. For GA-DPF, in each iteration, a total of $2M$ particles are used for exchange. Let L_g denotes the number of iterations, then the average communication complexity per agent is $\mathcal{O}(2L_gMd/K)$. Second, for computational complexity analysis, both GA-DPF and FA-DPF use GP-enhanced resampling in local particle filtering stage, and the complexity is $\mathcal{O}(M^3)$. And in this part, we exclude the filtering part and only focus on the comparison between two metaheuristic optimization algorithms. For genetic algorithm, the time complexity is usually $\mathcal{O}(Md)$, where M is the size of the population(i.e., the size of local particle set in our case) and d is the length of the genotypes(i.e., the dimension of the state vector in our case). For GA-DPF

under gossip protocol, the genetic algorithm is performed $2L_g$ times in total in each time duration, hence the average complexity for each agent is $\mathcal{O}(2L_gMd/K)$.

- GA-DPF(broadcast)
  If we use the GA-DPF under the local broadcast protocol, then in each communication iteration, there would be $2|E|M$ particles used for exchanging and the communication complexity per agent should be between $\mathcal{O}(2L_bMd)$ and $\mathcal{O}(2KL_bMd)$, where L_b is the number of broadcast iterations. For computational complexity, here the standard crossover operator is replaced by a multi-parent crossover, MFX. Let $D_{\max}$ denotes the maximum degree of the given network, and the complexity of the genetic algorithm becomes $\mathcal{O}(MdD_{\max})$. Since every agent needs to perform the GA L_b times per time duration, the total average computational complexity is $\mathcal{O}(L_bMdD_{\max})$.

- FA-DPF(gossip)
  The quantities exchanged in FA-DPF are slightly different from that in GA-DPF. In addition to exchanging particles, the likelihood values also need to be exchanged. Hence, in each iteration, a total of $2M$ particles as well as $4M$ likelihood values need to be exchanged. The average communication complexity per agent then should be $\mathcal{O}(L_gM(2d+4)/K)$. For computational complexity, the firefly algorithm requires $\mathcal{O}(Mlog(M)d)$ [60] since a sorting algorithm is needed in our implementation to seek "brighter" fireflies. And under the gossip protocol, the average complexity per agent should be $\mathcal{O}(2L_gMlog(M)d/K)$.

4.4.2 Discussion

In this Chapter, we apply the metaheuristic algorithms to the distributed particle filter to optimize the local particle set and achieve good performance. Compared with Chapter 3, the algorithms proposed in this Chapter further reduce the dependence on the size of the local particle set and have better performance, making it feasible to directly exchange particles in DPF. Next, we conduct a comparison between the algorithms proposed in this Chapter. Using GA-DPF under local broadcast communication conditions can obtain satisfactory tracking results with less communication overhead than under gossip communication conditions. However, if the communication overhead is further increased, under the condition of gossip, GA-DPF can further reduce the tracking error at a faster speed. While under the condition of local broadcast communication, the performance improvement of GA-DPF is limited. Therefore, we can conclude that, given a certain communication overhead, compared with multiple agents using the genetic algorithm for one-time information fusion, the operation of randomly selecting two agents for information fusion multiple times can approach the global optimal solution faster. And compared with using GA, using FA can greatly improve the tracking performance under low communication conditions. The disadvantage is that if the measurements only reflect a part of the dimensions of the state space, for the remaining part of the dimensions, as the number of iterations continues to increase, the uncertainty and error of the estimated value will also slightly increase.

4.4.3 Conclusion

The highlights of this Chapter are listed below.

- A genetic algorithm-assisted GP-DPF(GA-DPF) has been proposed under two communication conditions, gossip and local broadcast respectively. The main difference is the crossover operator we choose. Compared with the gossip communication condition, under the local broadcast condition, since each agent is able to communicate with multiple neighboring agents, we will have more than 2 "parents" when generating new offspring. Hence a multi-parent crossover operator is required. In this thesis, we use the MFX crossover operator.
- The simulation results show that the estimated performance will be improved whether using GA-DPF under the gossip condition or the local broadcast condition. Under local broadcast communication conditions, GA-DPF can obtain satisfactory tracking results with less communication overhead. In the case of gossip, although GA-DPF does not perform well with limited communication resources, if the communication overhead is further increased, the tracking error drops very fast. Numerical results verify the effectiveness of introducing the genetic algorithm.
- A firefly algorithm-assisted GP-DPF(FA-DPF) has been proposed under gossip communication condition. We omit the local broadcast condition here since there is little difference in implementing the FA-DPF in different communication conditions. Numerical results show that FA-DPF has very satisfactory performance in both low communication and high communication conditions, verifying the effectiveness of the firefly algorithm when being applied to DPF.
- A comparison of the computational and communication complexity of metaheuristic optimization-assisted GP-DPFs proposed in this Chapter has be given from a theoretical point of view. Besides, their limitations and challenges are also discussed.

5 Discussion

In this Chapter, we provide a comprehensive comparison and discussion of all the DPFs presented in this thesis. The following content will be presented from three dimensions, time, space, and communication complexity. To make it more compact, we use GP-DPF(consensus), GP-DPF(diffusive) to denote Improved GP-DPF(consensus-based) and Improved GP-DPF(diffusive mode) respectively in this Chapter.

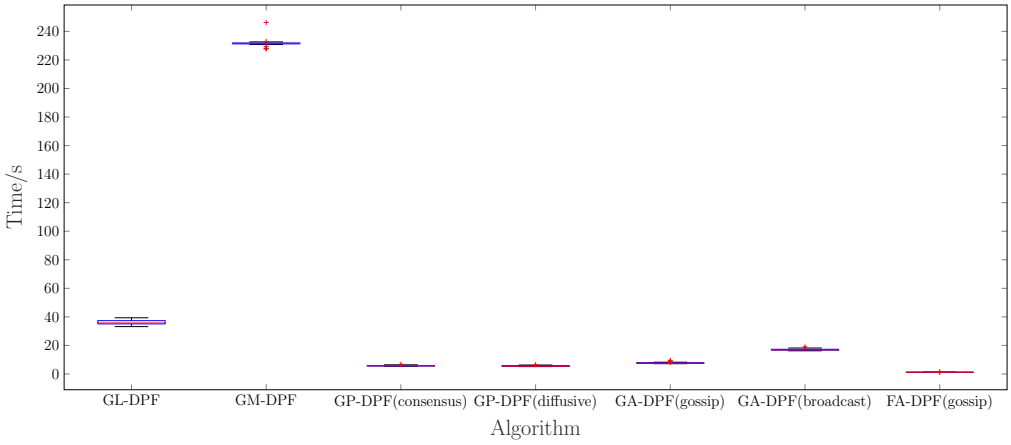
5.1 Time Complexity

First, we compare and analyze the time complexity. At the end of each previous Chapter, we have theoretically deduced the algorithms' time complexity, i.e., the computational complexity. Since it is hard to directly compare the time complexity of the presented algorithms from a theoretical perspective due to reason that the hyperparameters in each algorithm are mostly problem specific, in this Section, for a more intuitive display, we plot the running time required for different algorithms to achieve a given tracking error ARMSE (around 2), as shown in Fig 5.1. We conducted 50 Monte Carlo experiments for each algorithm and visualized the experimental results with box plots. From the figure we can see that the time complexity of the GM-DPF is much larger than that of other DPFs, and hence, it is difficult to apply to application scenarios with high real-time requirements. In contrast, the complexity of the GL-DPF algorithm is relatively much lower but still higher than all the DPF algorithms proposed in this thesis. In Fig 5.1(b), we compare all DPFs proposed in this thesis (note, the direct particle exchange based algorithm proposed in Section 3.3 is not documented due to its poor performance and failure to converge). Program runtimes vary from the shortest of about 1 second (FA-DPF) to the longest of 15 seconds (GA-DPF under broadcast manner).

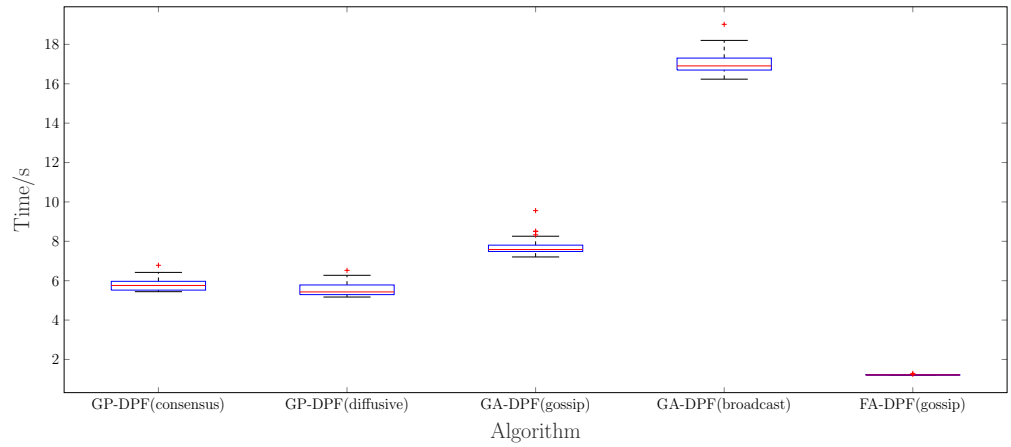
5.2 Space Complexity

Second, we analyze and compare the space complexity. Since the memory consumption of PF mainly depends on the size of the particle set, we use the particle set size as an indicator to measure the memory demand of each algorithm. In Fig 5.2, we plot the tracking performance of the algorithms proposed in Chapters 3 and 4 as a function of particle set size. Since all the DPFs proposed in this thesis use the GP-enhanced resampling algorithm, and it has been confirmed in Fig 3.1 that the GP-enhanced resampling algorithm greatly reduces the dependence on particle set size compared with the standard resampling algorithm, here we focus on the internal comparison between the algorithms proposed in this thesis. Fig 5.2 shows that although Algorithm 10 (GP-DPF based on consensus fusion) performs poorly at extremely low particle set

sizes, all 5 algorithms in the figure perform quite well at relatively low particle set sizes. At the same time, it can be seen that when metaheuristic optimization is used, the dependence of the particle filter on the size of the particle set is further reduced, but at the cost, the communication requirements of the algorithm are strengthened(i.e., more iterations are needed during a single time instant).



(a) including GL-DPF and GM-DPF.



(b) excluding GL-DPF and GM-DPF.

Figure 5.1: Time complexity comparison among all DPFs presented in this thesis.

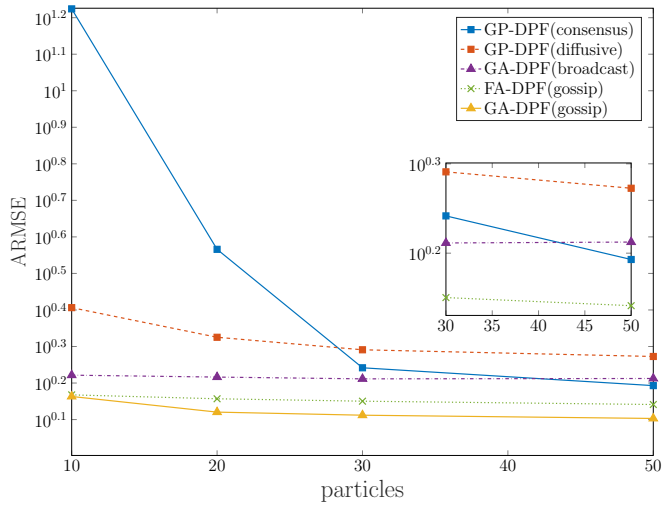


Figure 5.2: ARMSE as a function of varying particle size for all DPFs.

5.3 Communication overhead

Finally, we compare and analyze the communication resource requirements, which is also one of the most important metrics in distributed algorithms. The requirements for communication resources of different algorithms are usually affected by different multiple parameters. For example, in GM-DPF, the number of Gaussian components in the GMM and the number of iterations jointly determine the communication overhead. Since the communication overhead is determined by both the particle set size and the number of communication iterations in direct particle exchange based DPFs, when studying the relationship between the communication overhead and the estimation accuracy, we change the communication overhead by fixing the number of particles and changing the number of communication iterations. For each algorithm, we use the number of particles corresponding to its elbow point in Fig 5.2, i.e. 50 particles for GP-DPF (consensus), 50 particles for GP-DPF (diffusive), and 20 particles for GA-DPF (under gossip manner), 10 for GA-DPF (under broadcast manner) and 10 for FA-DPF (under gossip manner).

In Fig 5.3, we show the trajectory error ARMSE for each method as a function of the communication overhead. We quantify the communication overhead using the count of scalars transmitted between sensors in the network. As expected, there is a trade-off between estimation accuracy and communication overhead. From the figure, we can see that since GM-DPF parameterizes the posterior probability, it has the smallest communication overhead. While GL-DPF requires a higher communication overhead. The direct particle exchange based distributed particle filters proposed in this thesis can also obtain good estimation performance under limited communication resources. Algorithm 8 proposed in Section 3.3 has poor performance because it only randomly keeps the exchanged particles. The two improved versions proposed in Section 3.4 effectively overcome this limitation. However, we can see that the GP-DPF based

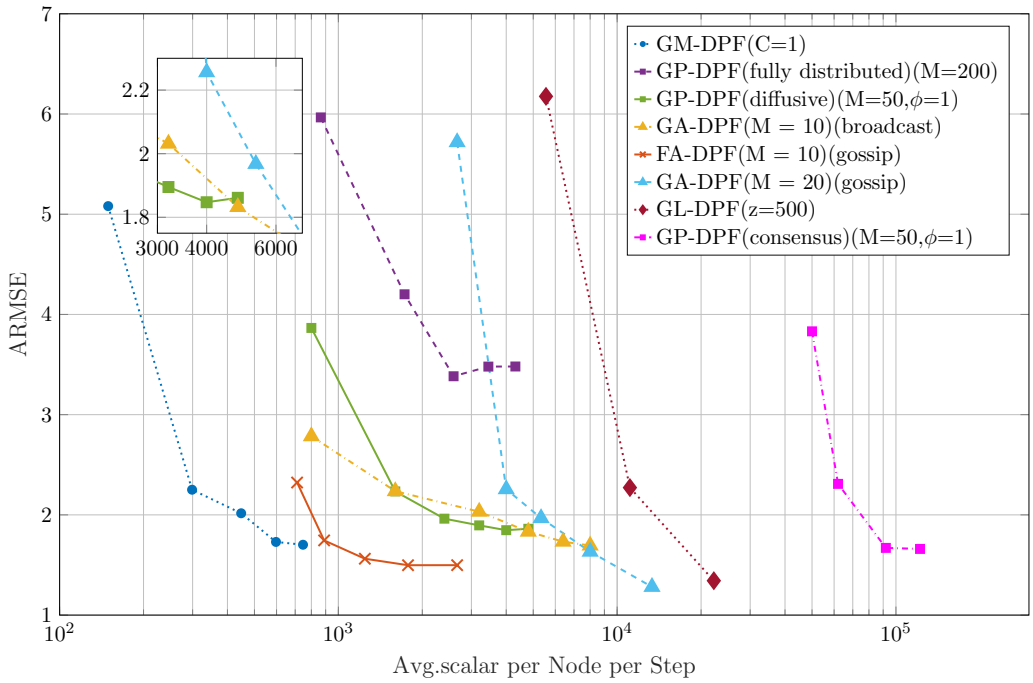


Figure 5.3: Trajectory estimation ARMSE as a function of the communication cost per time step.(x-axis in log scale).

on consensus fusion has a great demand for communication resources due to its need to reach a global consensus, while the algorithm based on diffusive mode can greatly reduce the communication overhead while maintaining good estimation performance. In Chapter 4, to seek the global optimal particle set at each moment, we introduced metaheuristic optimization and achieved good results. When adopting the genetic algorithm in DPF, we consider two communication architectures, namely gossip, and local broadcast. We find that under the gossip communication protocol, the uneven selection of agents will lead to insufficient global information fusion at the beginning, making the algorithm perform poorly in low traffic. However, when the traffic is further increased, the performance of the GA-DPF algorithm based on gossip communication is better than that based on local broadcast, which proves that the global optimal particle set can be approached more effectively by randomly selecting two agents and iterating for many times than by fusing the information of multiple agents at one time. Finally, compared with GA, using the firefly algorithm achieves a faster convergence rate in estimation error.

6 Conclusion and Future Work

6.1 Conclusion

This thesis attempts to study the distributed particle filter based on direct particle exchange. The proposed algorithms are all simulated in the scene of target tracking and the effectiveness of the algorithms is verified. The main contributions of this thesis are as follows.

- A resampling algorithm based on the Gaussian process is used for distributed particle filtering to reduce the dependence of the filter on the size of the particle set, which rationalizes the direct exchange of particles in low-dimensional state spaces. In Chapter 3, we perform information fusion based on both the consensus algorithm and diffusive mode and study the relationship between the performance and the communication resource consumption.

- Two metaheuristic optimization algorithms, the genetic algorithm, and the firefly algorithm are used to find the global optimal solution of the particle set in DPF. Simulations have shown the performance improvement of the combination.

- We comprehensively compare all the direct particle exchange based distributed particle filters proposed in this thesis with the state-of-the-art GL-DPF and GM-DPF from the three dimensions of time, space, and communication complexity, and present the necessary simulation results.

6.2 Future Work

The work of this thesis can be further studied and improved from the following aspects.

First, the DPFs proposed in this thesis are all based on the GP-enhanced resampling algorithm. However, in the current research, we also found that a large number of resampling algorithms also achieve the low dependence of particle filter on the particle set size, such as the resampling algorithm based on the genetic algorithm in [44]. These resampling algorithms can also be used in distributed scenarios and further studied.

Second, in Chapter 4, we use two metaheuristic optimization algorithms to search for the optimal particle set and obtain better estimation results. Likewise, more optimization algorithms [53] can be combined with direct particle exchange based distributed particle filtering to explore more possibilities.

Third, when applying the genetic algorithm under the local broadcast communication protocol in Section 4.2.3, for the crossover operator in the multi-parent scenario, we compared several existing algorithms and finally selected the MFX algorithm which

showed the best effect in the simulation part. In the future, crossover operators customized for distributed particle filter scenarios can be further studied.

Finally, the robustness of the proposed algorithms in harsh communication environments (e.g., link failures between nodes, communication delays, interruption in network connectivity) can also be further tested.

Bibliography

[1] F. Zhao, L. J. Guibas, and L. Guibas, *Wireless sensor networks: an information processing approach*. Morgan Kaufmann, 2004.

[2] F. Bullo, J. Cortés, and S. Martínez, "Distributed control of robotic networks: a mathematical approach to motion coordination algorithms," 2008.

[3] T. Shima and S. Rasmussen, *UAV cooperative decision and control: challenges and practical approaches*. SIAM, 2009.

[4] I. Levchenko, M. Keidar, J. Cantrell, Y.-L. Wu, H. Kuninaka, K. Bazaka, and S. Xu, "Explore space using swarms of tiny satellites," 2018.

[5] H. Aghajan and A. Cavallaro, *Multi-camera networks: principles and applications*. Academic press, 2009.

[6] O. Hlinka, F. Hlawatsch, and P. M. Djuric, "Distributed particle filtering in agent networks: A survey, classification, and comparison," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 61–81, 2012.

[7] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[8] T. Li, M. Bolic, and P. M. Djuric, "Resampling methods for particle filtering: classification, implementation, and strategies," *IEEE Signal processing magazine*, vol. 32, no. 3, pp. 70–86, 2015.

[9] Y. Bar-Shalom, P. K. Willett, and X. Tian, *Tracking and data fusion*, vol. 11. YBS publishing Storrs, CT, USA:, 2011.

[10] T. Imbiriba and P. Closas, "Enhancing particle filtering using gaussian processes," in *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pp. 1–7, IEEE, 2020.

[11] M. Rabbat, M. Coates, and S. Blouin, "Graph laplacian distributed particle filtering," in *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 1493–1497, IEEE, 2016.

[12] J. Li and A. Nehorai, "Distributed particle filtering via optimal fusion of gaussian mixtures," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 280–292, 2017.

[13] D. Üstebay, M. Coates, and M. Rabbat, "Distributed auxiliary particle filters using selective gossip," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3296–3299, IEEE, 2011.

[14] M. A. Vázquez and J. Míguez, "A robust scheme for distributed particle filtering in wireless sensors networks," *Signal Processing*, vol. 131, pp. 190–201, 2017.

[15] L. Martino, V. Elvira, and G. Camps-Valls, "Group importance sampling for particle filtering and mcmc," *Digital Signal Processing*, vol. 82, pp. 133–151, 2018.

[16] L. Martino and V. Elvira, "Compressed monte carlo with application in particle filtering," *Information Sciences*, vol. 553, pp. 331–352, 2021.

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

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

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

[20] A. Mohammadi and A. Asif, "Distributed particle filter implementation with intermittent/irregular consensus convergence," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2572–2587, 2013.

[21] D. Gu, "Distributed em algorithm for gaussian mixtures in sensor networks," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1154–1166, 2008.

[22] J. Li and A. Nehorai, "Adaptive gaussian mixture learning in distributed particle filtering," in *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pp. 221–224, IEEE, 2015.

[23] K. Dedecius and P. M. Djurić, "Sequential estimation and diffusion of information over networks: A bayesian approach with exponential family of distributions," *IEEE Transactions on Signal Processing*, vol. 65, no. 7, pp. 1795–1809, 2016.

[24] M. G. Bruno and S. S. Dias, "A bayesian interpretation of distributed diffusion filtering algorithms [lecture notes]," *IEEE Signal Processing Magazine*, vol. 35, no. 3, pp. 118–123, 2018.

[25] A. Mohammad and A. Asif, "Diffusive particle filtering for distributed multisensor estimation," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3801–3805, IEEE, 2016.

[26] W. Song, Z. Wang, J. Wang, F. E. Alsaadi, and J. Shan, "Distributed auxiliary particle filtering with diffusion strategy for target tracking: A dynamic event-triggered approach," *IEEE Transactions on Signal Processing*, vol. 69, pp. 328–340, 2020.

[27] O. Hlinka, F. Hlawatsch, and P. M. Djurić, "Distributed sequential estimation in asynchronous wireless sensor networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1965–1969, 2015.

[28] M. Li, W. Yi, and L. Kong, "A likelihood-based distributed particle filter for asynchronous sensor networks," in *2017 20th International Conference on Information Fusion (Fusion)*, pp. 1–5, IEEE, 2017.

[29] A. Mohammadi and A. Asif, "Distributed consensus + innovation particle filtering for bearing/range tracking with communication constraints," *IEEE Transactions on Signal Processing*, vol. 63, no. 3, pp. 620–635, 2014.

[30] A. Doucet, A. M. Johansen, *et al.*, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[31] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[32] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[33] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[34] F. Iutzeler, P. Ciblat, and J. Jakubowicz, "Analysis of max-consensus algorithms in wireless channels," *IEEE Transactions on Signal Processing*, vol. 60, no. 11, pp. 6103–6107, 2012.

[35] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[36] C. Rasmussen, "The infinite gaussian mixture model," *Advances in neural information processing systems*, vol. 12, 1999.

[37] G. J. McLachlan and S. Rathnayake, "On the number of components in a gaussian mixture model," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 341–355, 2014.

[38] K. Roeder and L. Wasserman, "Practical bayesian density estimation using mixtures of normals," *Journal of the American Statistical Association*, vol. 92, no. 439, pp. 894–902, 1997.

[39] A. Corduneanu and C. M. Bishop, "Variational bayesian model selection for mixture distributions," in *Artificial intelligence and Statistics*, vol. 2001, pp. 27–34, Morgan Kaufmann Waltham, MA, 2001.

[40] L. Xiao, S. Boyd, and S. Lall, "Distributed average consensus with time-varying metropolis weights," *Automatica*, vol. 1, 2006.

[41] J. R. Hershey and P. A. Olsen, "Approximating the kullback leibler divergence between gaussian mixture models," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 4, pp. IV–317, IEEE, 2007.

[42] S. Park, J. P. Hwang, E. Kim, and H.-J. Kang, "A new evolutionary particle filter for the prevention of sample impoverishment," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 801–809, 2009.

[43] S. S. Moghaddasi and N. Faraji, "A hybrid algorithm based on particle filter and genetic algorithm for target tracking," *Expert Systems with Applications*, vol. 147, p. 113188, 2020.

[44] N. Zhou, L. Lau, R. Bai, and T. Moore, "A genetic optimization resampling based particle filtering algorithm for indoor target tracking," *Remote Sensing*, vol. 13, no. 1, p. 132, 2021.

[45] M.-L. Gao, L.-L. Li, X.-M. Sun, L.-J. Yin, H.-T. Li, and D.-S. Luo, "Firefly algorithm (fa) based particle filter method for visual tracking," *Optik*, vol. 126, no. 18, pp. 1705–1711, 2015.

[46] M. Ebden, "Gaussian processes: A quick introduction," *arXiv preprint arXiv:1505.02965*, 2015.

[47] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018.

[48] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994.

[49] D.-T. Nguyen, M. Filippone, and P. Michiardi, "Exact gaussian process regression with distributed computations," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 1286–1295, 2019.

[50] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[51] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *International symposium on stochastic algorithms*, pp. 169–178, Springer, 2009.

[52] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," *Computational intelligence for multimedia big data on the cloud with engineering applications*, pp. 185–231, 2018.

[53] X.-S. Yang, "Metaheuristic optimization," *Scholarpedia*, vol. 6, no. 8, p. 11472, 2011.

[54] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[55] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[56] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995.

[57] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[58] S. Tsutsui and A. Ghosh, "A study on the effect of multi-parent recombination in real coded genetic algorithms," in *1998 IEEE international conference on evolutionary computation proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pp. 828–833, IEEE, 1998.

[59] S. Tsutsui, M. Yamamura, and T. Higuchi, "Multi-parent recombination with simplex crossover in real coded genetic algorithms," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pp. 657–664, 1999.

[60] X.-S. Yang and X. He, "Firefly algorithm: recent advances and applications," *International journal of swarm intelligence*, vol. 1, no. 1, pp. 36–50, 2013.