A
Dissertation
On

Analysis of Sensor Fusion using Kalman Filter in Homogeneous and Heterogeneous Swarm Networks

Submitted in Partial Fulfillment of the Requirement

For the Award of the Degree of

Master of Technology

In

Computer Science & Engineering

Submitted By

Puneet Jain
University Roll No. 2K11/CSE/09

Under the Esteemed Guidance of
Mr. Vinod Kumar
Assoc. Prof., Computer Engineering Department, DTU, Delhi

2011-2013

DELHI TECHNOLOGICAL UNIVERSITY

DELHI - 110042

CERTIFICATE

This is to certify that the dissertation titled "**Analysis of Sensor Fusion using Kalman Filter in Homogeneous and Heterogeneous Swarm Networks**" is a bonafide record of work done at **Delhi Technological University** by **Puneet Jain, Roll No. 2K11/CSE/09** for partial fulfilment of the requirements for the degree of Master of Technology in Computer Science & Engineering. This project was carried out under my supervision and has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

(Mr. Vinod Kumar)

Assoc. Professor & Project Guide

Date: __________ **Department of Computer Engineering**

Delhi Technological University

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all the people who have supported and encouraged me during the course of this project without which, this work could not have been accomplished.

First of all, I am very grateful to my project supervisor Mr. Vinod Kumar for providing the opportunity of carrying out this project under his guidance. I am deeply indebted to him for the support, advice and encouragement he provided without which the project could not have been a success. I am also grateful to Dr. (Mrs.) Daya Gupta, HOD, Computer Science, DTU for her immense support. I am also thankful to my parents for being there for me at all times. Last but not the least; I am grateful to Delhi Technological University for providing the right resources and environment for this work to be carried out.

Puneet Jain
University Roll no: 2K11/CSE/09
M.Tech (Computer Science & Engineering)
Department of Computer Engineering
Delhi Technological University
Delhi – 110042

ABSTRACT

A swarm consists of simplistic mobile nodes sharing information to achieve a common complex task. Most real world sensors result in noisy data which must be corrected to achieve accurate results. Noise filter or estimation algorithms are very useful in achieving this task. Sensor fusion is another useful technique to attain more useful and meaningful data from multiple sources of less accurate data. In the proposed system a group of mobile nodes are used to measure a changing parameter of another underlying system. But the measurement taken by these nodes is assumed to be erroneous. Thus, an instance of the standard discrete Kalman filter is used at each node to obtain a more accurate estimate. In this dissertation, two sharing algorithms are proposed for homogeneous and heterogeneous swarms. Each algorithm uses two different types of sharing schemes, complex and simple. A simple analysis of the standard Kalman filter is conducted to observe its characteristics. The comparative analysis of the two proposed algorithms and a standard moving average shows improved performance of both algorithms. Also, the behaviour of the algorithms is studied with change in number of member nodes, sensor range, communication range, sensor accuracy and type of sharing in the system. It is also shown that the heterogeneous swarm performs better with the proposed heterogeneous algorithm than the homogeneous algorithm which does not account for the difference in each node.

Table of Contents

List of Figures

1. Introduction

Distributed estimation is a fundamental information processing problem in swarm networks. A swarm system or swarm network consists of a group of mobile nodes communicating and sharing information with each other to achieve a common goal. A real world swarm network would considerably depend on accurate estimation and sharing of noisy data. All solutions to such a problem must take into account the dynamic nature of swarm networks.

1.1 Problem statement

In a given system of a group of nodes (swarm), a common dynamic parameter of an underlying system may be measured. Each node of the swarm independently measures the value of that parameter, along with certain error and disseminates the information in the whole swarm. The goal is to estimate the accurate real time value of the parameter.

1.2 Related topics

A swarm network may be broadly classified as homogeneous and heterogeneous. Homogeneous swarms consist of identical nodes, whereas, heterogeneous swarms consist of dissimilar nodes. Both swarms systems must be analyzed separately. Sensor fusion is the task of integrating data from multiple sources resulting in more meaningful or useful data. Sensor fusion has wide application in distributed environment. But sensor fusion is not effective if data is noisy. Hence, accurate estimation of data is very important. Noise filters are an essential class of filters in the field of signal processing. Kalman filter is one such noise filter. The standard Kalman filter consists of a set of mathematical equations that provides an efficient computational method to estimate the state of a given process, in a way that minimizes the mean of the squared error [13]. The filter is simple and powerful in the sense that it only requires the previous state to estimate the next state of a process. A Kalman filter with non uniform sampling is used to work effectively with the distributed and dynamic swarm.

1.3 Proposed work

In this dissertation, two sensor fusion algorithms are proposed for both homogeneous and heterogeneous swarm networks. The homogeneous system consists of nodes having identical sensors with similar error magnitude and on the other hand, the heterogeneous system consists of nodes having dissimilar sensors and error magnitude. The Kalman filter is utilized to accurately estimate the noisy values measured by the sensors. Also, identical and dissimilar filters operate on each node of homogeneous and heterogeneous system respectively. The behaviour of both systems in different conditions is analyzed.

The behaviour of a standard Kalman filter is analyzed for reference. A comparative analysis of homogeneous and heterogeneous swarms, along with respective algorithms, is conducted. The analysis based on different number of nodes clearly shows improved performance for both algorithms compared to a standard moving average. It also shows improved performance of the heterogeneous algorithm. Also, the behaviour of the heterogeneous swarm with the homogeneous algorithm is analyzed to understand the behaviour of the heterogeneous system with both algorithms. It is clearly observed that the heterogeneous system shows increased performance with its corresponding algorithm.

2. Swarms

A swarm system or a swarm network consists of a group of mobile nodes (identical or different) communicating and sharing information to achieve a common goal. The ‘swarm’ intelligence is characterized by cooperative or collective behaviour of very simplistic nodes with limited capabilities to achieve complex global intelligence.

Some natural examples of swarm intelligence are bees and ants colonies, bacterial growth, animal herding, bird flocking and fish schooling. Several swarm algorithms have been developed by emulating such natural behaviour[14,15,29,32,33]. Individual organisms living in large groups naturally follow simple rules to achieve a larger goal for the whole group. Flocking and schooling are easily achieved by the participating individuals by simply observing their neighbours and correcting their own position with respect to them, this leads to elaborate and complex group motion. Insects such as bees and ants largely rely on swarm behaviour. Tasks such as foraging, colony defense, exploration etc. are direct results of complex swarm behaviour arising from simple individual behaviour and communication among the group.

The swarm system is largely decentralized and thus, does not completely rely on specific decision making or control nodes. Therefore, all algorithms adapted for swarm system must also be decentralized. Some swarms may be completely decentralized and rely on individual decision making using data gathered from neighbours. Other types of systems depend on master-slave or clustering (with a chosen cluster head) configurations. In such cases, a representative node is chosen for either the whole swarm or a sub cluster of the swarm. All or partial decision making is done by this representative node. Algorithms for these systems must handle failure of representative node, reselection of representative node, manage communication between the representative node and other members [4,20,28,31]. These decentralized nodes also have self organizing capability. This supports dynamic increase or decrease in number of nodes actively participating in the swarm. Also, the roles performed by these nodes may change over a period of time. This fluctuation in the number of nodes and their roles must be managed and handled properly since it is an essential feature of all swarms. The inherent decentralized and self organizing characteristics lead to a very adaptive and dynamic system. Such a system has a very high fault tolerance and shows increased efficacy in adapting to changing and unpredictable future states.

All swarms systems rely on effective communication among the nodes. This dynamic network of such mobile nodes is essential for swarm intelligence. Members of a swarm constantly share their position and other problem specific information. This shared information is the basis for all higher level algorithms. To properly function most swarming algorithms also require that all nodes be synchronized. Such synchronization can easily be achieved by using standard algorithms such as the well known Lamport's logical clock [17,28].

A network of nodes exhibiting swarm intelligence may be of two types, homogeneous swarms consist of identical member nodes and heterogeneous swarms. and heterogeneous swarms consist of dissimilar nodes.

2.1 Homogeneous swarms

Such swarms are more common in theoretical study [2,5,25]. Standard study of swarm intelligence is based on nodes with identical attributes and capabilities. All member nodes exhibit identical motion characteristics, have identical sensors, identical processing capabilities, identical communication and other data collection and processing standards. Such swarms are simpler to manage due to the inherent redundancy and equality among the nodes. A failure or addition of a node can be easily handled.

2.2 Heterogeneous swarms

Such swarms are less frequently studied but are essential in real world applications [8,18,21]. Heterogeneous swarms are more versatile since they take into account the slight difference in attributes of each node. Such swarms categorize nodes into different classes. This categorization may be based on several factors such as agility, efficiency of computation, efficiency of available sensors etc. The algorithms utilized in such swarms are more complex since they must handle different types of nodes and each node may or may not have another equivalent node. Heterogeneous swarms allow incorporation of multiple nodes with a large number of differences, for example, a single cohesive swarm may consist of a few slow moving nodes with very large processing capability, and a large number of small, simplistic and fast moving nodes with limited capabilities.

3. Sensor Fusion

Sensor fusion is the process of integration of data obtained from multiple identical or dissimilar sensors measuring a common or disparate source, such that the resulting information is more consistent, accurate and useful [3,9]. Simplest sensor fusion may have completely identical sensors measuring a common source. A complex system may consist of multiple dissimilar sensors measuring multiple different sources to obtain more accurate state metrics of the environment being observed. It is well studied and very essential in distributed systems such as senor networks and swarms [3,6,9,16].

Sensor fusion is of two type centralized and decentralized, depending on where the fusion occurs. In centralized fusion, all data is sent to a central location where it is processed and fused. In a decentralized system, the data may be processed by multiple individual sites and processing data independently. An extremely dynamic system such as a swarm, usually utilizes decentralized sensor fusion. Each node processes data obtained by multiple neighbours. Though, there is no single site dependency in such systems, the fused sensor data at multiple sites may or may not be same due to dependency on available neighbours and different sensor fusion process states at different sites.

Real world data usually has a component of unwanted error in it. Reduction of this error in measurement of a single error prone source can be achieved in different ways; using multiple different sensors measuring simultaneously, or identical sensors measuring continuously over a period of time, or a combination or the two methods. Sensor fusion can be applied effectively to integrate the data measured by multiple sensors to estimate an accurate value of the source. Sensor fusion may also be used to gather measurement from multiple sources and use their data to estimate the value of a related parameter.

Sensor fusion largely depends on and is a combination of statistical analysis, filtering and estimation techniques. Fusion techniques utilized over a period of time may gradually estimate a more accurate value or may be used to continuously measure a source. Thus, in dynamic systems or networks such as swarms, synchronization is essential among member nodes [17].

4. Noise Filter

Noise filters or estimators are essential in statistical analysis and signal processing. Measurement of most real world sources under observation or study lead to erroneous data. If the original source model is known then most noise can easily be filtered out. But most estimators gradually filter the noise over a period of time utilizing past data and an approximate model of the original source under study. The most basic estimator is a simple average of all past measurements or a moving average of past *k* measurements. But most commonly used filters are more complex and take into account the approximate model of the source to estimate a future value.

Bayesian estimators are very common class of noise filters which are based on Bayesian statistics. It is a subfield of statistics in which evidence about true state of the system under study is expressed in terms of "Degree of belief" or Bayesian probability. Bayes' theorem, eq. (4.1), links the degree of belief in a proposition before and after accounting for evidence.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.1}$$

For proposition A and evidence B,

- $P(A)$, the prior, is the initial degree of belief in A.
- $P(A|B)$, the posterior, is the degree of belief having accounted for B.
- The quotient $\frac{P(B|A)}{P(B)}$ represents the support B provides for A.

Thus, in such estimators the priori data is used to estimate a more accurate posterior data.

4.1 Wiener Filter

The wiener filter was proposed by Norbert Wiener during 1940's [30]. This filter assumes that one has prior knowledge about both signal and the (additive) noise, which are stationary linear stochastic processes. This priori information is used by this linear time-invariant filter to obtain a more accurate posterior. The wiener filter is based on the Minimum Mean-Square Error (MMSE) estimator, which minimizes the mean square error in a Bayesian setting.

4.2 Particle Filter

The particle filter, also known as Sequential Monte Carlo method (SMC), is a complex estimation technique which is based on simulation [1,7]. The particle filter uses Monte Carlo method, i.e., it does not use a closed form expression but instead uses repeated random sampling to obtain results. The random samples or particles are obtained in a given probability distribution. The various results obtained from different samples are compared and a few more accurate samples are chosen. This step is repeated until a tighter set of samples is obtained. Unlike standard Kalman and Wiener filter, there are very few assumptions and limitations. The system may or may not be linear and noise may or may not be Gaussian. Though particle filters are versatile and fast, but they are also very complex and may not be effectively utilized by very simplistic member nodes of a swarm. To use such a filter each node would have to run its own complex particle filter due to decentralized sensor fusion.

4.3 Kalman Filter

In 1960, R.E. Kalman published his famous paper describing a solution to the discrete-data linear filtering problem [13].The Kalman filter is a set of mathematical equations that provides an efficient computational means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown. The Kalman filter has been the subject of extensive research and application [22,23,24,26,27].

4.3.1 Standard filter with uniform sampling

The standard discrete Kalman filter is based on the assumption that the underlying system has a linear evaluation function and all noise in the system is additive and follows normal or Gaussian distribution.

The Kalman filter estimates the state $x \in \mathbb{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_k = A_k x_{k-1} + B_k u_{k-1} + w_{k-1}, \tag{4.2}$$

With a measurement $y \in \mathbb{R}^m$ that is

$$y_k = B_k x_k + v_k. \tag{4.3}$$

The $n \times n$ matrix A_k in eq. (4.2) relates the state at previous time step $k - 1$ to the state at current time step k. The $n \times l$ matrix B_k relates the optional control input $u \in \mathbb{R}^l$ to the state x. The m× n matrix H_k in the measurement eq. (4.3) relates the state to the measurement y_k. The random variables w_k and v_k represent the process and measurement noise respectively. They are assumed to be independent of each other, white, and with normal probability distributions. Due to uniform sampling, time differences between consecutive iterations are constant. And matrices A, B and H may or may not be dependent on the time (or iteration).

The Kalman filter algorithm has two phases, *time update* and *measurement update*.

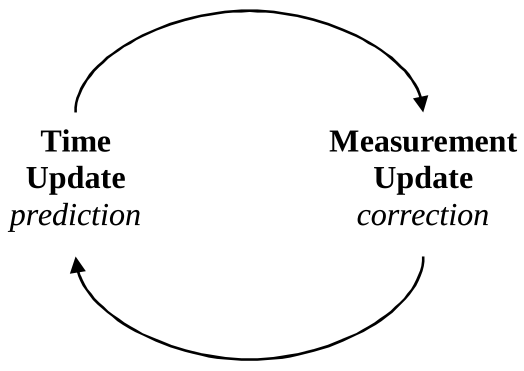


Figure 4.1 Discrete Kalman filter cycle

The *time update* step is projects the current state estimate ahead in time. The *measurement update* adjusts the projected estimated by an actual measurement that time.

The *time update* is described by following equations,

$$\hat{x}_k^- = A_k\hat{x}_{k-1} + B_k u_{k-1}, \tag{4.4}$$

$$P_k^- = A_k P_{k-1} A_k{}^T + Q_k. \tag{4.5}$$

The *measurement update* is described by the following equations,

$$K_k = P_k^- H_k{}^T (H_k P_k^- H_k{}^T + R_k)^{-1}, \tag{4.6}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-), \tag{4.7}$$

$$P_k = (I - K_k H_k)P_k^-. \tag{4.8}$$

The variables in eq. (4.4-4.8) are,

$\hat{x}_k$	: Estimated state, posteriori.
$\hat{x}_k^-$	: Estimated state, priori.
A_k	: State transition matrix (i.e., transition between states).
u_k	: Control variable.
B_k	: Control matrix (i.e., mapping control to state variables).
P_k	: State variance matrix (i.e., error of estimation), posterior.
P_k^-	: State variance matrix (i.e., error of estimation), priori.
Q_k	: Process variance matrix (i.e., error due to process; covariance of $w_k$from eq. (4.2)).
y_k	: Measurement variables.
H_k	: Measurement matrix (i.e., mapping measurements onto state).
K_k	: Kalman gain.

R_k : Measurement variance matrix (i.e., error from measurements; covariance of v_k from eq. (4.3)).

In the *time update* step, eq. (4.4) and eq. (4.5) project state and covariance estimates from step $k-1$ to step k. A_k and B_k are obtained from eq. (4.2).

The first step in the *measurement update* step is to calculate the Kalman gain. In eq. (4.6) it can be seen that as measurement error covariance R_k approaches 0, the gain K_k weights the residual more heavily.

$$\lim_{R_k \to 0} K_k = H_k^{-1} \tag{4.9}$$

On the other hand, as the *priori* estimate error covariance approaches zero, the gain K_k weights the residual less heavily.

$$\lim_{P_k^- \to 0} K_k = 0 \tag{4.10}$$

Depending on the initial variable values, the Kalman filter may require a few iterations to stabilize and get a tighter estimate.

4.3.2 Filter with non uniform sampling

A slight modification of the Kalman filter with uniform sampling is the non uniform sampling filter [19]. The original filter is based on constant time difference between iterations. But applications such as distributed analysis and swarm networks, such time synchronization is not possible. Therefore, another variant of the Kalman filter utilizes iteration steps with different time intervals.

Similar to the standard filter, the equations are:
The *time update* is described by following equations,

$$\hat{x}_{k,t_2}^- = A_{\Delta t}\hat{x}_{k-1,t_1} + B_{\Delta t}u_{\Delta t}, \tag{4.11}$$

$$P^{-}_{k,t_2} = A_{\Delta t} P_{k-1,t_1} {A_{\Delta t}}^{T} + Q_{\Delta t}. \tag{4.12}$$

The *measurement update* is described by the following equations,

$$K_{k,t_2} = P^{-}_{k,t_2} {H_{t_2}}^{T} (H_{t_2} P^{-}_{k,t_2} {H_{t_2}}^{T} + R_{t_2})^{-1}, \tag{4.13}$$

$$\hat{x}_{k,t_2} = \hat{x}^{-}_{k,t_2} + K_{k,t_2}(y_{k,t_2} - H_{t_2}\hat{x}^{-}_{k,t_2}), \tag{4.14}$$

$$P_{k,t_2} = (I - K_{k,t_2} H_{t_2}) P^{-}_{k,t_2}. \tag{4.15}$$

In eq. (4.11) and eq. (4.12) A, B, u and Q depend on $\Delta t = t_2 - t_1$. In eq. (4.13-4.15) H and R only depend on t_2 because they are independent of the previous state.

This filter can be effectively applied in a swarm system where measurement of parameters at uniform time intervals may not be possible. Thus, each measurement at a different time may be processed along with previous such measurements.

4.3.3 Other versions

Kalman filter has been extensively researched and modified [11,12]. Well known variants such as the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) provide wider applications and improved capabilities[10]. The EKF is a nonlinear version of the standard Kalman filter and unlike the standard filter does not require only additive noise. In highly nonlinear state transition and observation models the EKF may give poor results because the covariance is propagated through linearization of the underlying nonlinear model. The UKF utilizes a deterministic sampling technique known as unscented transform to choose a minimal set of sample points around the mean. These points are propagated through the nonlinear functions, from which the mean and covariance of the estimate are then recovered. This resulting filter more accurately captures the actual mean and covariance.

5. Sensor Fusion in Swarms using Kalman Filter

Sensor fusion using Kalman filter in a distributed system has only been studied by a few [23,24,26]. The Kalman filter and its variants can be easily applied in a dynamic and distributed setting. Important criteria for any information sharing algorithm are, when to share, which neighbours to share with and what to share. Sharing algorithms must account for dissimilar states of different nodes. Such algorithms must balance the need for sharing of complex large amounts of data and, energy consumption and delay.

In the proposed sensor fusion system, the Kalman filter with non uniform sampling is applied in a distributed swarm system consisting of multiple, mobile, identical or dissimilar nodes. The nodes may measure a single or multiple parameters of a system. Each node may run a single or multiple instances of the sharing algorithm depending on observation of single or multiple parameters, respectively. Each node is assumed to have a single type of sensor, but the proposed algorithms can easily be adapted for multisensory nodes. All nodes are assumed to be synchronized. Each node has a sensor range and a communication range. The sensor range is the distance up to which the sensor can effectively measure an observation point. The communication range is the distance up to which the node can communicate with another member node.

In the proposed system, there are two different algorithms for homogeneous and heterogeneous swarms. Three metrics are used to identify how and what to share. Firstly, the total number of iterations of Kalman filter executed to obtain current estimated value known to node i, N_i. This helps in identifying the stability achieved by the filter. Secondly, the time stamp of the latest measurement known to node i, Tl_i. It is important to known that it is not necessary that node i measure and execute the filter itself. It may receive estimates and measurements from other nodes. Lastly, the average rank of the estimated value known to node i, R_i. This is further explained in section 5.2 and is only applicable for heterogeneous swarms.

Each algorithm uses two different types of sharing methods. One is a simple sharing method which involves direct transfer of data from one node to another. The other is a more complex version which involves multiple interchange and processing of data between the two nodes. These are referred to as ShareSimple and ShareComplex methods, respectively.

5.1 Homogeneous swarm with Kalman filter

Homogeneous swarms consist of identical nodes with identical sensors accuracy, sensor range, motion characteristics, and processing capability. Data obtained by all nodes is treated equally. Two control parameters are used to minutely control how and when information is shared. They are stability number, $NSTAB$ and time difference, $TDIFF$. It is more preferable that an estimator captures the true value as soon as possible. Thus, as number of filter iterations increases a tighter estimate is obtained compared to initial values. A balance between the number of iterations and more recent estimates must be made.

The simple sharing method followed by nodes i and j belonging to a homogeneous swarm is given as:

$\mathbf{ShareSimpleHo(i, j)}$

- Send Tl_i, N_i, $\hat{x}_{N_i,Tl_i}$, P_{N_i,Tl_i} from i to j.
- Send $y_{m,t}$ for $m = N_i - k$ to $m = N_i$, i.e. send last k measurements from i to j.
- Node j updates its variables.

In this method, node i sends all relevant information it has to node j. The number of measurements to shared may be fixed at the start, given by k. The las k t measurements may be used by j when it might execute a ShareComplexHo method. Thus, updating node j.

The complex sharing method followed by nodes i and j belonging to a homogeneous swarm is given as:

$\mathbf{ShareComplexHo(i, j)}$

- Send Tl_j, $\hat{x}_{N_j,Tl_j}$, P_{N_j,Tl_j} from j to i.

- Using $\hat{x}_{N_j,Tl_j}$ and P_{N_j,Tl_j} node i iterates Kalman filter from its measurement value with next greater time index to Tl_j up to last measurement value. Note that $Tl_i > Tl_j$.
- Last $k - c$ measurement values are sent from node j to i and are added in sorted order according to time. Here, c is the number of measurement values of i which were iterated over in the above step.
- Node i shifts last k measurement values by c.
- Node i updates $N_i = N_j + c$. Thus, obtaining new $\hat{x}_{N_i,Tl_i}$ and P_{N_i,Tl_i}.
- Send Tl_i, N_i, $\hat{x}_{N_i,Tl_i}$, P_{N_i,Tl_i} from i to j.
- Send $y_{m,t}$ for $m = N_i - k$ to $m = N_i$, i.e send last k measurements from i to j.
- Node j updates its variables.

The complex sharing method ShareComplexHo(i, j) has three phases. In the first phase, node j sends its estimate, state variance and last measurement time to node i. In the second phase, node i iterates up to its last measurement value using the data given by j. Node j also sends additional measurement values to have accurate last k measurements at node i. Node i then updates its own measurement list and other variables. In phase three, node i performs steps similar to simple sharing and sends data to node j.

The complex sharing method is explained in Figure 5.1. In this figure $k = 6$ and due to values of Tl_i and Tl_j, $c = 4$. First node j sends data to i, then i processes and updates its data, then it shares this new updated values with node j.

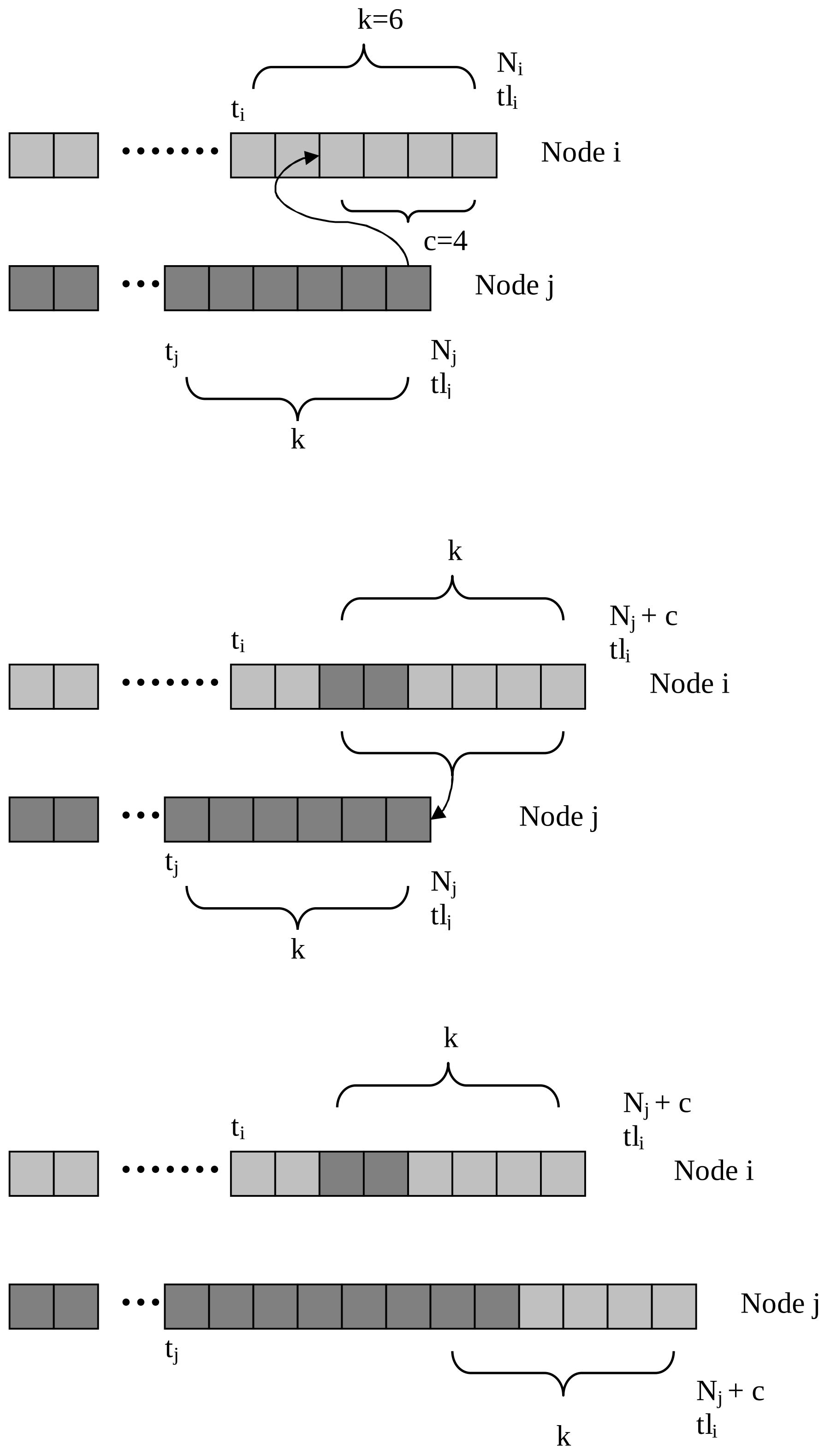


Figure 5.1 Complex sharing method for node *i* to *j*.

The sharing algorithm followed by two neighbor nodes i and j for a single observation source is given as:

Homogeneous Sharing Algorithm

```
If $N_i$ == $N_j$
    If $N_i$ <= $NSTAB$
        If $Tl_i$ > $Tl_j$
            ShareComplexHo(i, j)
        Else If $Tl_i$ < $Tl_j$
            ShareComplexHo(j, i)

    Else if $Tl_i$ > $Tl_j$
        If $Tl_i - Tl_j > TDIFF$
            ShareSimpleHo(i, j)
        Else
            ShareComplexHo(i, j)

    Else if $Tl_i$ < $Tl_j$
        If $Tl_j - Tl_i > TDIFF$
            ShareSimpleHo(j, i)
        Else
            ShareComplexHo(j, i)
Else if $N_i$ > $N_j$
    If $N_i$ <= $NSTAB$
        If $Tl_i$ > $Tl_j$
            ShareComplexHo(i, j)
        Else if $Tl_i$ < $Tl_j$
            ShareComplexHo(j, i)
        Else if $Tl_i$ == $Tl_j$
            ShareSimpleHo(i, j)
    Else if $Tl_i$ > $Tl_j$
        ShareSimpleHo(i, j)
        Else if $Tl_i$ < $Tl_j$
            If $N_j$ > $NSTAB$
                ShareSimpleHo(j, i)
            Else
                ShareComplexHo(j, i)
        Else if $Tl_i$ == $Tl_j$
            ShareSimpleHo(i, j)
    Else if $N_j$ > $N_i$
        If $N_j$ <= $NSTAB$
            If $Tl_j$ > $Tl_i$
                ShareComplexHo(j, i)
            Else if $Tl_j$ < $Tl_i$
                ShareComplexHo(i, j)
            Else if $Tl_j$ == $Tl_i$
                ShareSimpleHo(j, i)
        Else if $Tl_j$ > $Tl_i$
            ShareSimpleHo(j, i)
        Else if $Tl_j$ < $Tl_i$
            If $N_i$ > $NSTAB$
                ShareSimpleHo(i, j)
            Else
                ShareComplexHo(i, j)
        Else If $Tl_j$ == $Tl_i$
            ShareSimpleHo(j, i)
```

In general, nodes with larger number of known measurements and/or more recent measurement are used as source nodes to send distribute data to other nodes. During the starting phase of the system it is desirable that more sharing take place. This enables the filter to stabilize faster. $NSTAB$ is used to control the number of iterations up to which combination of measured values and re-estimation is required. After number of iterations is greater than $NSTAB$ the node with latest measurement sends data to other nodes.

If N_i == N_j then $TDIFF$may be used to control if re-estimation is required or not since. The complex sharing method enables combination of multiple measurements and results in single more accurate estimated value.

5.2 Heterogeneous swarm with Kalman filter

In the heterogeneous swarm all nodes are not identical. In such a swarm nodes having varying capabilities. Specifically, some nodes have more accurate sensors and longer sensor range. Thus, nodes have different v_k and R_k.

In the proposed scheme, a rank or score is assigned to each type of sensor. Higher rank/score implies more accurate sensor. If a single node measures the observation point then its estimated value will have the rank same as it sensor rank. But in the case of swarms where nodes share measurement values and average rank of the estimated value must be obtained.

The simple sharing of data from node i to j, where both belong to heterogeneous swarm is given as:

$\mathbf{ShareSimpleHe(i, j)}$

- Send Tl_i, N_i, $\hat{x}_{N_i,Tl_i}$, P_{N_i,Tl_i}, AR_i from i to j.
- Send $y_{m,t}$ for m = N_i – k to m = N_i, i.e. send last k measurements from i to j.
- Node j updates its variables.

The only difference between homogeneous and heterogeneous is the additional sharing of the average rank of estimated value at node i is AR_i.

The complex sharing of data from node i to j, where both belong to heterogeneous swarm is given as:

$\mathbf{ShareComplexHe(i, j)}$

- Send Tl_j, $\hat{x}_{N_j,Tl_j}$, P_{N_j,Tl_j} from j to i.

- Using $\hat{x}_{N_j,Tl_j}$ and P_{N_j,Tl_j} node i iterates Kalman filter from its measurement value with next greater time index to Tl_j up to last measurement value. Note that $Tl_i > Tl_j$.
- Last $k - c$ measurement values are sent from node j to i and are added in sorted order according to time. Here, c is the number of measurement values of i which were iterated over in the above step.
- Node i shifts last k measurement values by c.
- Node i updates $N_i = N_j + c$. Thus, obtaining new $\hat{x}_{N_i,Tl_i}$ and P_{N_i,Tl_i}.
- Node i also updates AR_i given as:

$$AR_i = \frac{AR_j * N_j + AR_i * c}{N_j + c}. \tag{5.1}$$

- Send Tl_i, N_i, $\hat{x}_{N_i,Tl_i}$, P_{N_i,Tl_i}, AR_i from i to j.
- Send $y_{m,t}$ for $m = N_i - k$ to $m = N_i$, i.e send last k measurements from i to j.
- Node j updates its variables.

The complex sharing method of heterogeneous is also different from homogeneous only with respect to additional update and sharing of AR_i.

The sharing algorithm followed by two neighbor nodes i and j for a single observation source is given as:

Heterogeneous Sharing Algorithm

```
$AR_i == AR_j$
    If $N_i == N_j$
        If $N_i <= NSTAB$
            If $Tl_i > Tl_j$
                ShareComplexHe(i, j)
            Else if $Tl_i < Tl_j$
                ShareComplexHe(j, i)
        Else if $Tl_i > Tl_j$
        If $Tl_i - Tl_j > TDIFF$
            ShareSimpleHe(i, j)
        Else
            ShareComplexHe(i, j)
    Else if $Tl_i < Tl_j$
        If $Tl_j - Tl_i > TDIFF$
            ShareSimpleHe(j, i)
        Else
            ShareComplexHe(j, i)
Else if $N_i > N_j$
    If $N_i <= NSTAB$
        If $Tl_i > Tl_j$
            ShareComplexHe(i, j)
        Else if $Tl_i < Tl_j$
            ShareComplexHe(j, i)
        Else if $Tl_i == Tl_j$
            ShareSimpleHe(i, j)
    Else if $Tl_i > Tl_j$
        ShareSimpleHe(i, j)
    Else if $Tl_i < Tl_j$
        If $N_j > NSTAB$
            ShareSimpleHe(j, i)
        Else
            ShareComplexHe(j, i)
    Else if $Tl_i == Tl_j$
        ShareSimpleHe(i, j)
Else if $N_i > N_i$
    If $N_j <= NSTAB$
        If $Tl_j > Tl_i$
            ShareComplexHe(j, i)
        Else if $Tl_j < Tl_i$
            ShareComplexHe(i, j)
            Else if $Tl_j == Tl_i$
                ShareSimpleHe(j, i)
        Else if $Tl_j > Tl_i$
            ShareSimpleHe(j, i)
        Else if $Tl_j < Tl_i$
            If $N_i > NSTAB$
                ShareSimpleHe(i, j)
            Else
                ShareComplexHe(i, j)
        Else If $Tl_j == Tl_i$
            ShareSimpleHe(j, i)
$AR_i > AR_j$
    If $N_i == N_j$
        If $N_i == NSTAB$
            If $Tl_i > Tl_j$
                ShareComplexHe(i, j)
            Else if $Tl_i < Tl_j$
                ShareComplexHe(j, i)
            Else if $Tl_i == Tl_j$
                ShareSimpleHe(i, j)
        Else if $Tl_i > Tl_j$
            If $Tl_i - Tl_j > TDIFF$
                ShareSimpleHe(i, j)
            Else
                ShareComplexHe(i, j)
        Else if $Tl_i < Tl_j$
            If $Tl_j - Tl_i > TDIFF * Rf$
                ShareSimpleHe(j, i)
            Else
                ShareComplexHe(j, i)
        Else if $Tl_i == Tl_j$
            ShareSimpleHe(i, j)
```

```
Else if $N_i > N_j$
    If $N_i <= NSTAB$
        If $Tl_i > Tl_j$
            ShareComplexHe(i, j)
        Else if $Tl_i < Tl_j$
            ShareComplexHe(j, i)
        Else if $Tl_i == Tl_j$
            ShareSimpleHe(i, j)
    Else if $Tl_i > Tl_j$
        ShareSimpleHe(i, j)
    Else if $Tl_i == Tl_j$
        ShareSimpleHe(i, j)
    Else if $Tl_j - Tl_i > TDIFF * Rf$
        ShareSimpleHe(j, i)
    Else
        ShareComplexHe(j, i)
Else if $N_j > N_i$
    If $N_j <= NSTAB$
        If $Tl_j > Tl_i$
            ShareComplexHe(j, i)
        Else if $Tl_j < Tl_i$
            ShareComplexHe(i, j)
        Else if $Tl_j == Tl_i$
            ShareSimpleHe(j, i)
    Else if $Tl_j > Tl_i$
        ShareSimpleHe(j, i)
    Else If $Tl_j == Tl_i$
        ShareSimpleHe(j, i)
    Else if $Tl_i - Tl_j > TDIFF$
        ShareSimpleHe(i, j)
    Else
        ShareComplexHe(i, j)
```

```
$AR_i < AR_j$
    If $N_j == N_i$
        If $N_j <= NSTAB$
            If $Tl_j > Tl_i$
                ShareComplexHe(j, i)
            Else if $Tl_j < Tl_i$
                ShareComplexHe(i, j)
            Else if $Tl_j == Tl_i$
                ShareSimpleHe(j, i)
        Else if $Tl_j > Tl_i$
            If $Tl_j - Tl_i > TDIFF$
                ShareSimpleHe(j, i)
            Else
                ShareComplexHe(j, i)
        Else if $Tl_j < Tl_i$
            If $Tl_i - Tl_j > TDIFF * Rf$
                ShareSimpleHe(i, j)
            Else
                ShareComplexHe(i, j)
        Else if $Tl_j == Tl_i$
            ShareSimpleHe(j, i)
    Else if $N_j > N_i$
        If $N_j <= NSTAB$
            If $Tl_j > Tl_i$
                ShareComplexHe(j, i)
            Else if $Tl_j < Tl_i$
                ShareComplexHe(i, j)
            Else if $Tl_j == Tl_i$
                ShareSimpleHe(j, i)
        Else if $Tl_j > Tl_i$
            ShareSimpleHe(j, i)
        Else if $Tl_j == Tl_i$
```

```
                ShareSimpleHe(j, i)
            Else if $Tl_i - Tl_j > TDIFF * Rf$
                ShareSimpleHe(i, j)
            Else
                ShareComplexHe(i, j)
        Else if $N_i > N_j$
            If $N_i <= NSTAB$
                If $Tl_i > Tl_j$
                    ShareComplexHe(i, j)
                Else if $Tl_i < Tl_j$
                    ShareComplexHe(j.i)
                Else if $Tl_i == Tl_j$
                    ShareSimpleHe(i, j)
            Else if $Tl_i > Tl_j$
                ShareSimpleHe(i, j)
            Else If $Tl_i > Tl_j$
                ShareSimpleHe(i, j)
            Else if $Tl_j - Tl_i > TDIFF$
                ShareSimpleHe(j, i)
            Else
                ShareComplexHe(j, i)
```

The main difference between homogeneous and heterogeneous sharing algorithms is the average rank which must also be compared. The accuracy of sensor is essential to obtain a more accurate estimate. But in some cases, for example, if a high ranking node has a very old measurement value, it would not be preferable for such a node to share its data, a less accurate but more recent measurement may be more useful. This balance must be maintained. A fixed rank factor, Rf, is used to control the balance between accurate measurements and more recent measurements. The positive Rf is multiplied with $TDIFF$ to control when complex sharing takes place. Complex sharing allows more accurate data to be included in estimates.

6. Simulation Results

The simulation study has been conducted using MATLAB 7.12.0 (R2011a) and no additional packages were used. In this study all results have been obtained independent of the network. That is, it is assumed that no communication delays, protocol delays, synchronization delays, and interference due to communication signal occur. It is also assumed that no processing delays take place.

The motion of the swarm occurs in a Euclidean space. A fixed range is selected in which the nodes are allowed to move. A fixed maximum velocity is chosen and the velocity of each node is randomly selected and follows uniform distribution.

The simulation environment consists of a single observation point (green) and multiple mobile nodes (magenta). At the observation point temperature is said to be increasing following the eq. (6.1), where A and B are fixed parameters. All noise in the system follows normal or Gaussian distribution. All operations are independent of units of measurement.

$$Temp_{new} = A * Temp_{current} + B \tag{6.1}$$

6.1 Standard Kalman Filter

In this study a 50 by 50 area is considered along with a single stationary node constantly monitoring the observation point. The increasing temperature is described as $A = 1.03$ and $B = 1.5$ from eq. (6.1), with an initial temperature of 100.

Figure 6.1 describes the standard Kalman filter with parameters as, Measurement noise magnitude (standard deviation) = 45, $Inital\ \hat{x} = 0$, $Initial\ P = 100$, $Q = 10$, $H = 1$, $F = 1.03$, $B = 1.5$ and $u = 1$. It can be seen that F and B are accurate but Q indicates that node assumes process error. A moving average with past 5 values is also considered for comparative analysis.

It can be seen in Figure 6.1, that the estimated value is more accurate than the moving average. Also, that the Kalman filter captures a tighter estimate after some time. This is due to the fact that the initial estimate is very far from actual value.

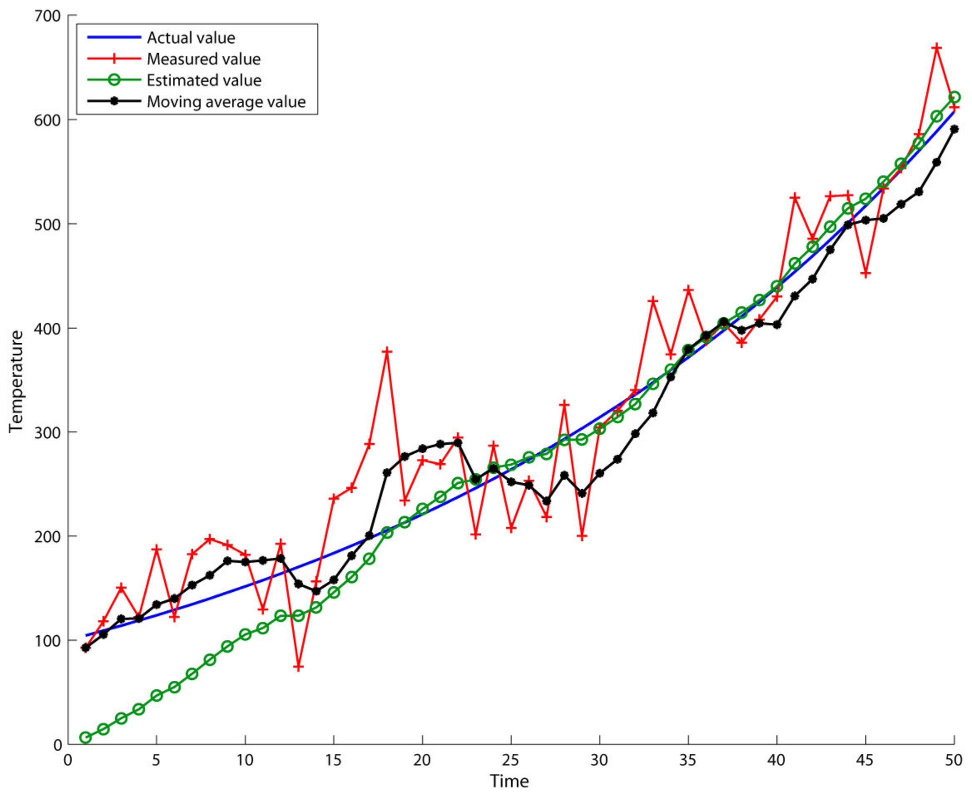


Figure 6.1 Standard Kalman filter P=100 and Q=10

In Figure 6.2, it is considered that $Q = 50$ and all other parameters are same. Thus, there is deviation in the estimated value since the measuring node assumes that the system model parameters F and B are not accurate. Therefore, some more time is taken by the filter capture the tighter values.

In Figure 6.3, it is considered that $P = 1000$ and all other parameters are same. Since, the filter assumes large error in the initial estimate, it captures the true value much faster. But such a large error takes time to reduce, so there is deviation from actual value in the beginning.

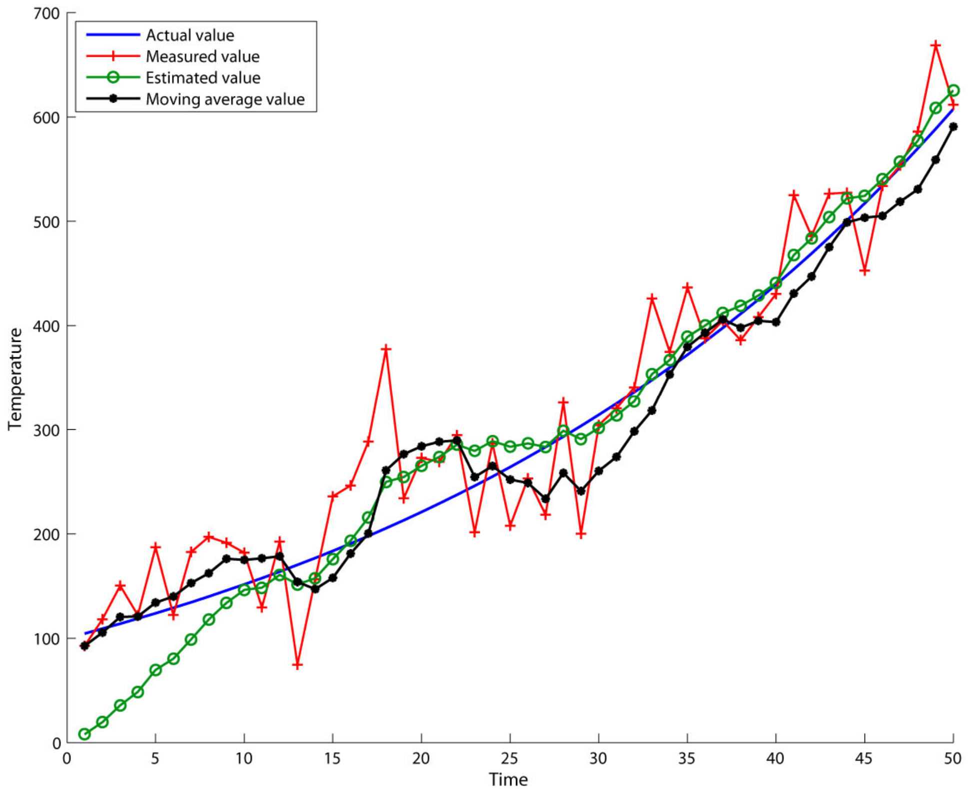


Figure 6.2 Standard Kalman filter P=100 and Q=50

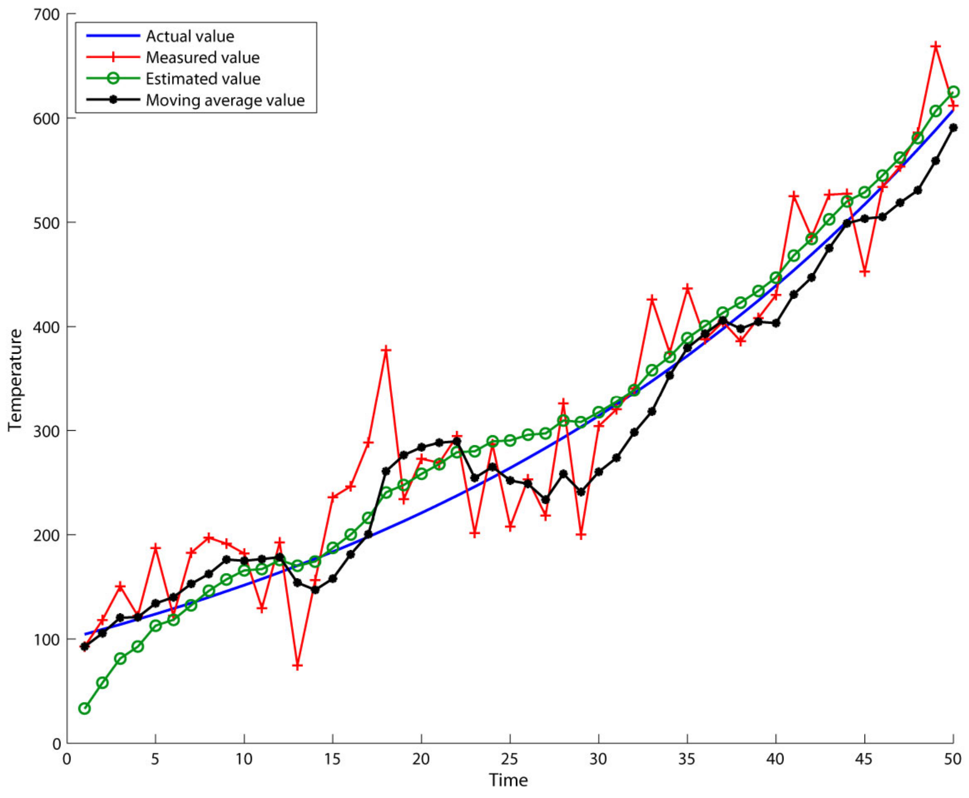


Figure 6.3 Standard Kalman filter P=1000 and Q=10

6.2 Homogeneous swarm algorithm with Kalman filter

Similar to the standard Kalman filter model, a single observation point is assumed but with multiple identical nodes. The temperature is said to be increasing with A = 1.09 and B= 1, from eq. (6.1), with starting temperature of 100.

The non uniform sampling Kalman filter parameters are, Measurement noise magnitude = 150, $Inital\,\hat{x}$ = 0 , $Initial\,P$ = 500, Q = 10, H = 1, u = 1, and F and B are accurate. Eq. (6.1) is fixed time interval increase of temperature but nodes measure only when they are close to observation point. Therefore, F and B must be adapted according to eq. (6.2) and eq. (6.3).

$$F_{\Delta t} = 1.019^{\Delta t} \tag{6.2}$$

$$B_{\Delta t} = \frac{1.019^{\Delta t} - 1}{1.019 - 1} \tag{6.3}$$

The swarm parameters are, $No.\,of\ nodes$ = 5, $Maximum\ velocity$ = 20, $Communication\ range$ = 30, $Sensor\ range$ = 10.

Figure 6.4 describes the modelled environment with the observation point and mobile nodes along with their communication and sensor ranges. In figure 6.5 $NSTAB$ = 0 and $TDIFF$ = 10. It can be seen that the estimated value is not close to actual. Large fluctuation in the average measured shows reduced sharing and higher error. At many occasions it can be seen that both estimates and measured are horizontal to x axis, thus showing no sharing. In figure 6.6 $NSTAB$ = 10 and $TDIFF$ = 10, thus increasing the complex sharing. A slight improvement in estimated values is observed compared to previous run.

To comparatively study the effect of number of nodes, two models of 10 nodes and 30 nodes were used, as described in figure 6.7 and figure 6.8 respectively. In both cases the temperature increase model is same as earlier. The non uniform sampling Kalman filter parameters are same as earlier, i.e. Measurement noise magnitude = 150, $Inital\,\hat{x}$ = 0

, $Initial\ P$ = 500, Q = 10, H = 1, u = 1, and F and B are given by eq. (6.2) and eq. (6.3) respectively.

In figure 6.9, the swarm parameters are, $No.of\ nodes$ = 10, $Maximum\ velocity$ = 15, $Communication\ range$ = 15, $Sensor\ range$ = 10, $NSTAB$ = 5, $TDIFF$ = 5. Figure 6.10 has same parameters but $No.of\ nodes$ = 30. It can be clearly seen that increase in number of nodes provides a more accurate estimate. Also, the effectiveness of the algorithm is highly dependent on the frequency of sharing and observation. Due to reduced frequency of sharing and observation and high error the moving average gives closer measurement values. As iterations increase the estimated value attains a tighter capture of the actual value.

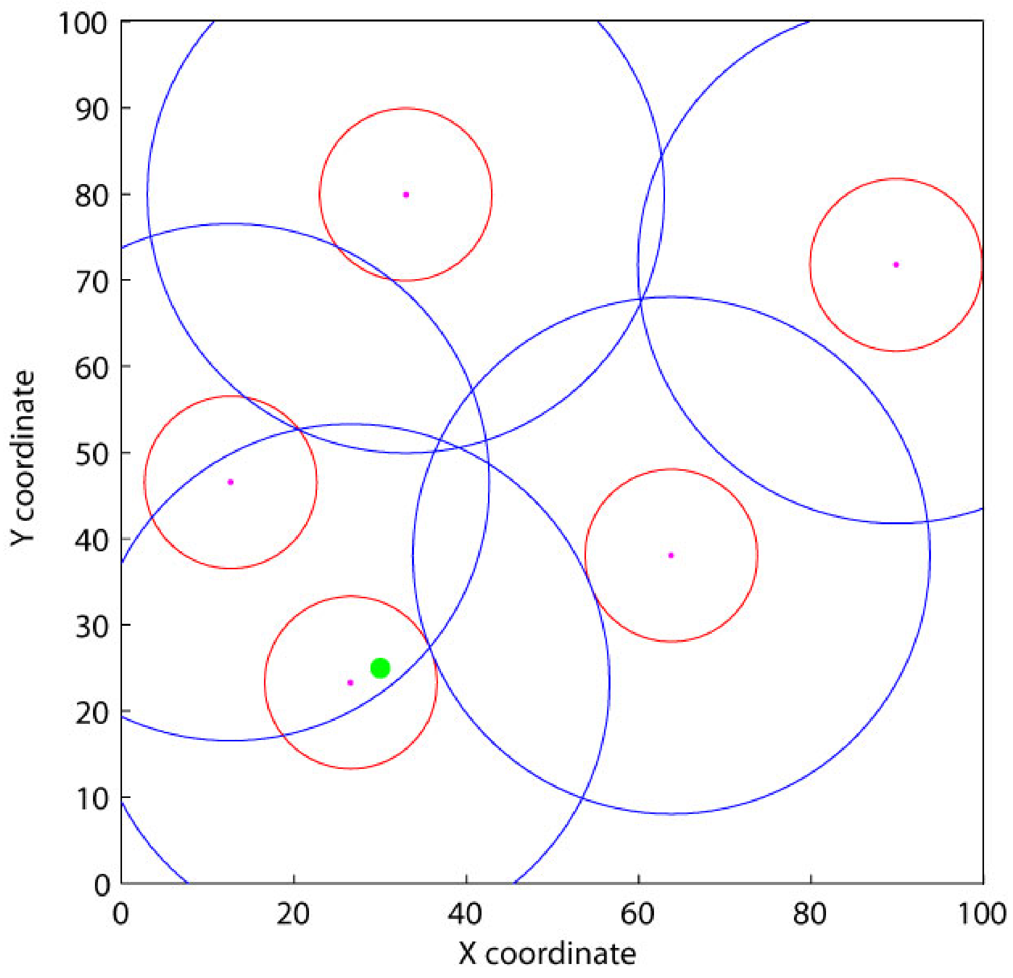


Figure 6.4 Model with observation point (green), 5 nodes (magenta), comm. range (blue)=30 and sensor range (red)=10.

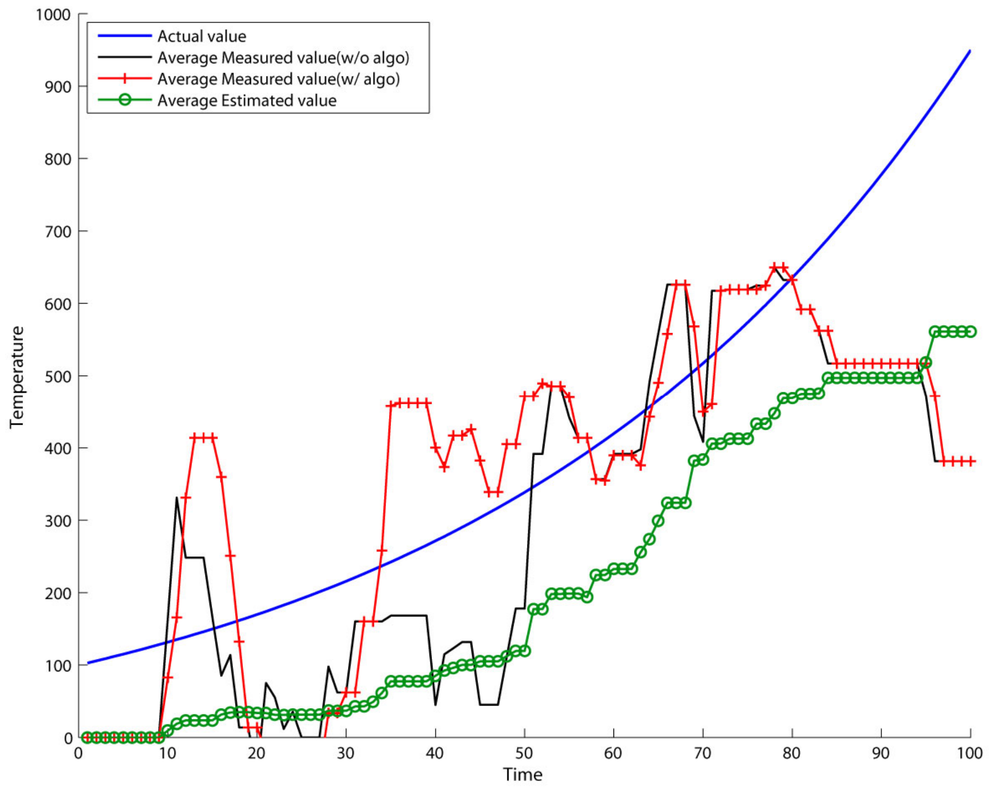


Figure 6.5 Homogeneous 5 nodes with NSTAB=0 and TDIFF=10.

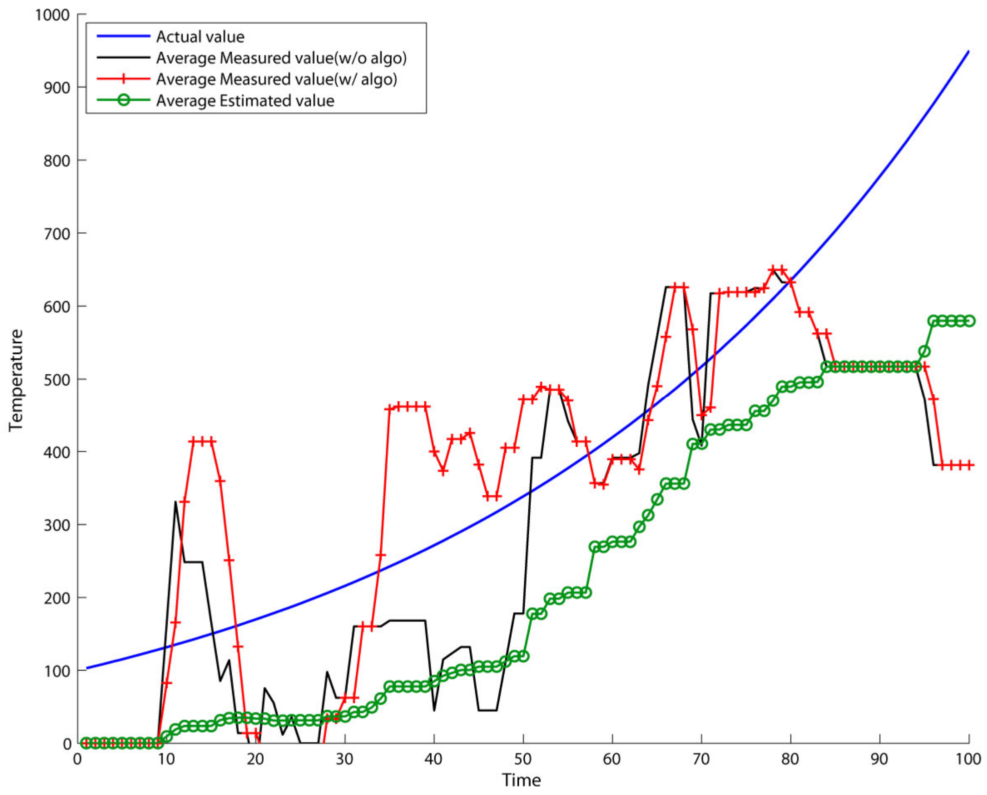


Figure 6.6 Homogeneous 5 nodes with NSTAB=10 and TDIFF=10.

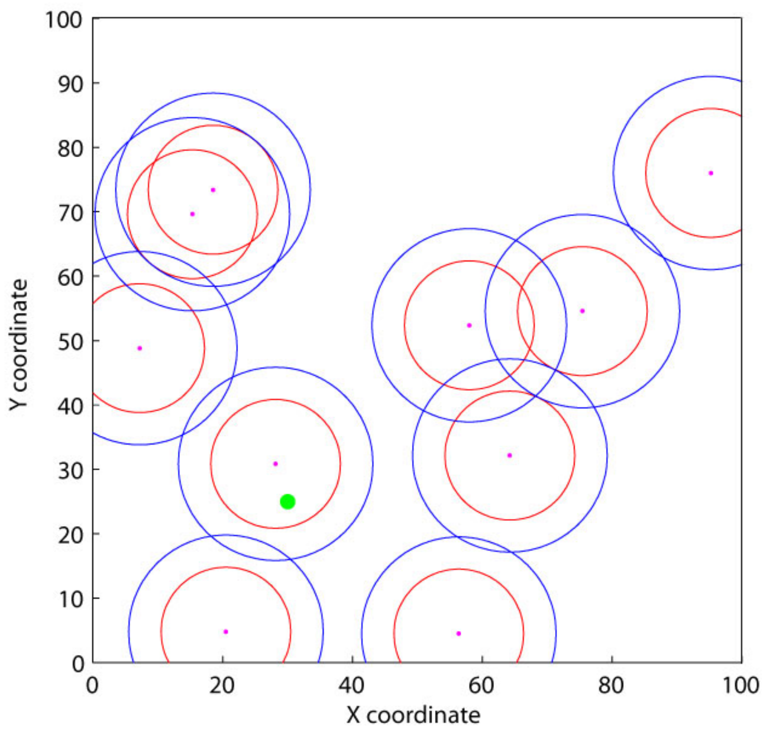


Figure 6.7 Model with observation point (green), 10 nodes (magenta), comm. range (blue)=15 and sensor range (red)=10.

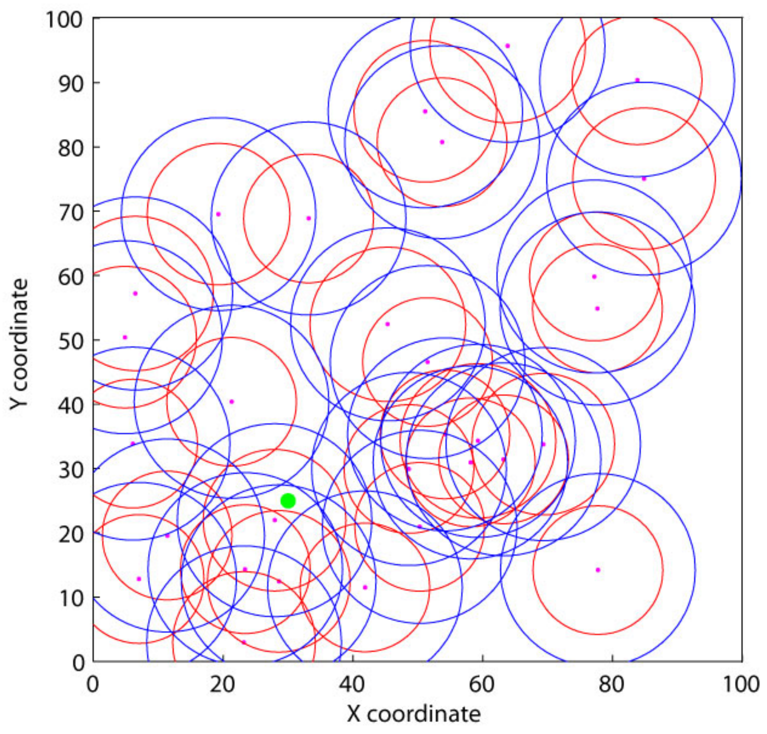


Figure 6.8 Model with observation point (green), 30 nodes (magenta), comm. range (blue)=15 and sensor range (red)=10.

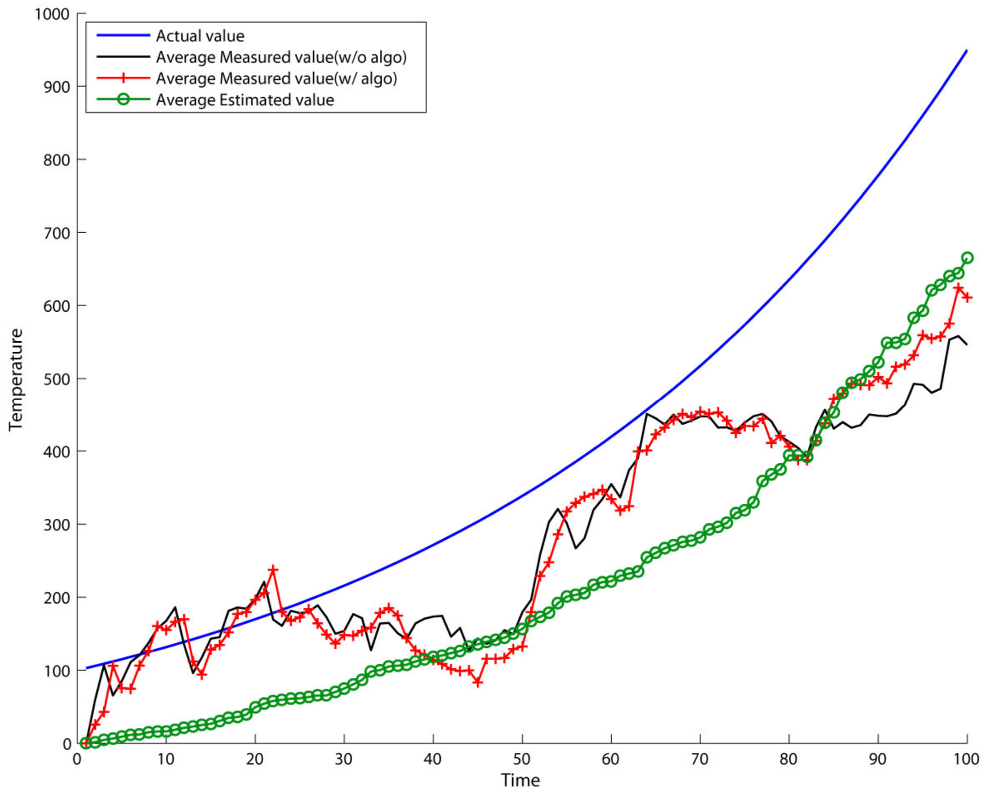


Figure 6.9 Homogeneous swarm and algorithm with 10 nodes.

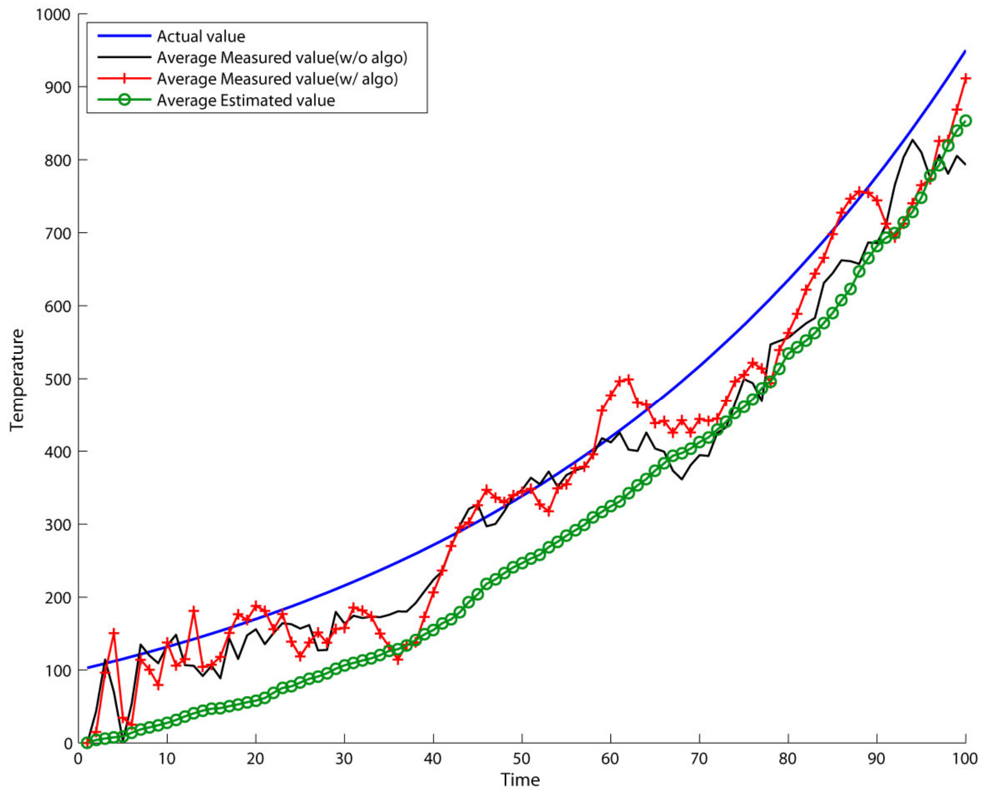


Figure 6.10 Homogeneous swarm and algorithm with 30 nodes.

6.3 Heterogeneous swarm algorithm with Kalman filter

In the heterogeneous swarm model some nodes have more accurate sensors and greater sensor range. Figure 6.11 and figure 6.12 describe models with Rf = 1.5 and Rf = 2, respectively. The increasing temperature model is same as above with A = 1.019 and B = 1 in eq. (6.1), with initial temperature of 100. The non uniform sampling Kalman filter parameters are same as earlier, i.e. $Inital\ \hat{x}$ = 0 , $Initial\ P$ = 500, Q = 10, H = 1, u = 1, and F and B are given by eq. (6.2) and eq. (6.3) respectively. The swarm parameters are, $No.of\ nodes$ = 30, $Maximum\ velocity$ = 15, $Communication\ range$ = 15, $NSTAB$ = 5, $TDIFF$ = 5. But unlike homogeneous swarms these nodes are not identical, 18 nodes have Measurement noise magnitude = 150 and $Sensor\ range$ = 10, 8 nodes have Measurement noise magnitude = 80 and $Sensor\ range$ = 11, and 4 nodes have Measurement noise magnitude = 5 and $Sensor\ range$ = 12.

It can be clearly seen in figure 6.11 and figure 6.12 that a much tighter capture is attained. Also, around time=80, Rf = 1.5 causes less dip is estimated value and Rf = 2 causes more. Rf must be used to obtain a fine balance between recent and more accurate measurements. Larger Rf means more preference to accurate measurements than recent measurements.

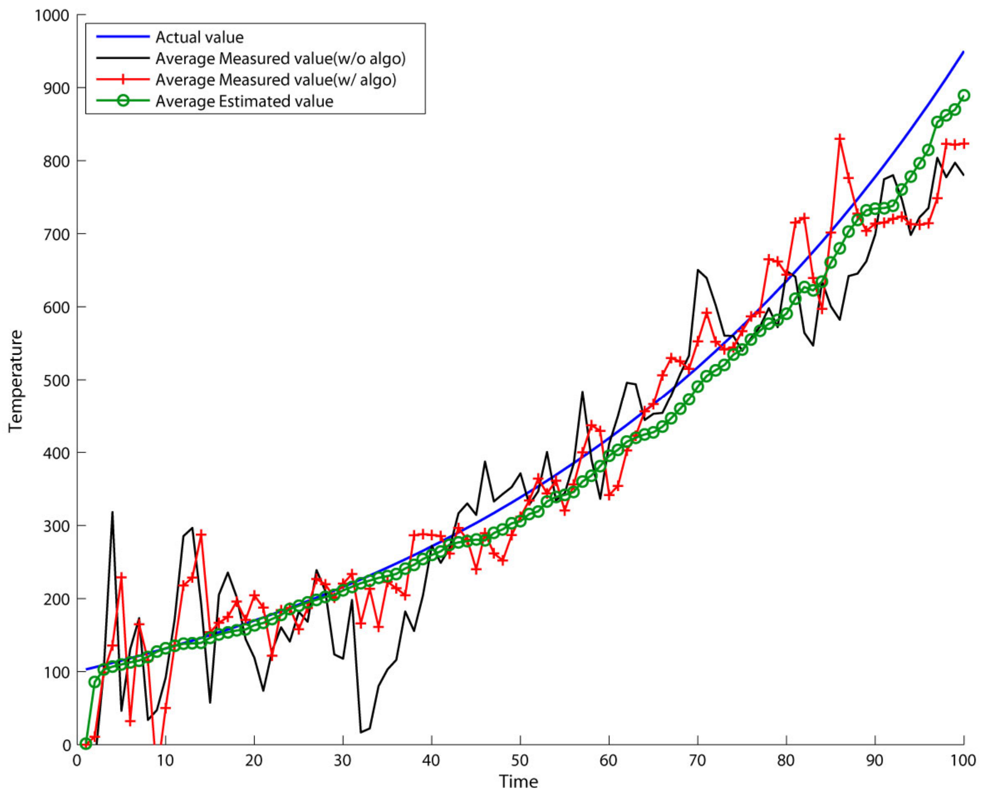
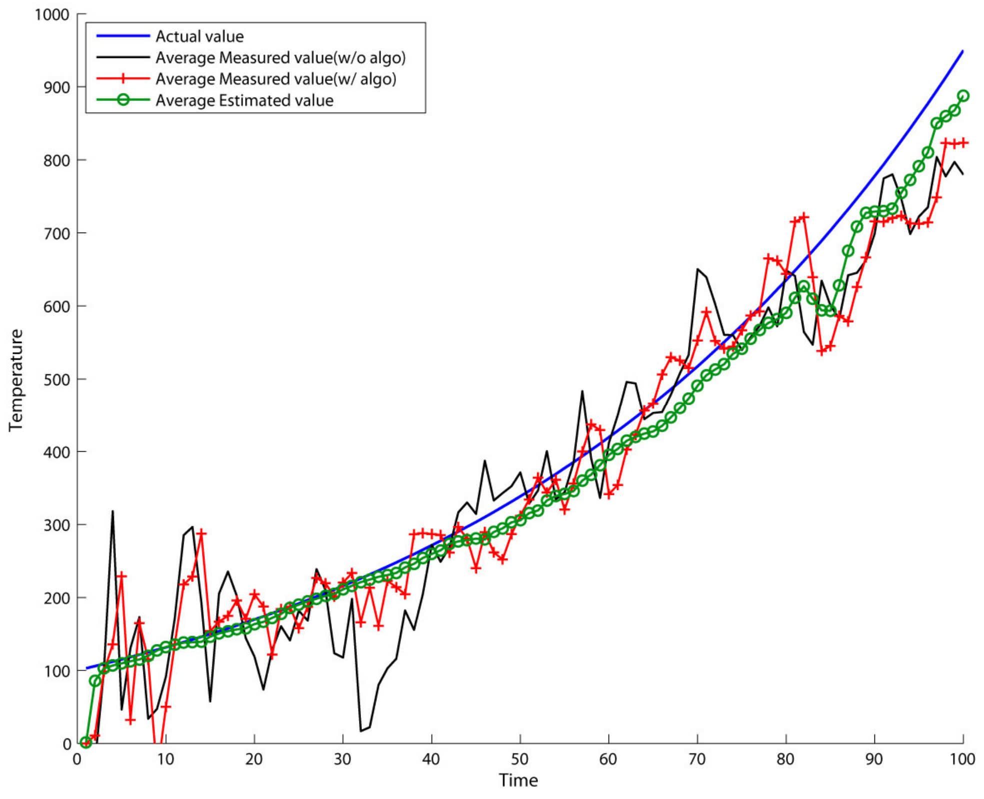


Figure 6.11 Heterogeneous swarm and algorithm with 30 nodes Rf=1.5.



i

Figure 6.12 Heterogeneous swarm and algorithm with 30 nodes Rf=2.

In figure 6.13 and figure 6.14 we compare the effect of number of nodes on this algorithm. Same filter setting are used as above. But the swarm in figure 6.13 has 10 nodes; 5 nodes have Measurement noise magnitude = 150 and $Sensor\ range$ = 10, 3 nodes have Measurement noise magnitude = 80 and $Sensor\ range$ = 11, and 2 nodes have Measurement noise magnitude = 25 and $Sensor\ range$ = 12. Figure 6.14 has 30 nodes; 18 nodes have Measurement noise magnitude = 150 and $Sensor\ range$ = 10, 8 nodes have Measurement noise magnitude = 80 and $Sensor\ range$ = 11, and 4 nodes have Measurement noise magnitude = 5 and $Sensor\ range$ = 12. An obvious increase in accuracy of estimation is observed. This is due to presence of more accurate nodes as well as increased sharing among the nodes. Also, clear improvement from the moving average is seen.

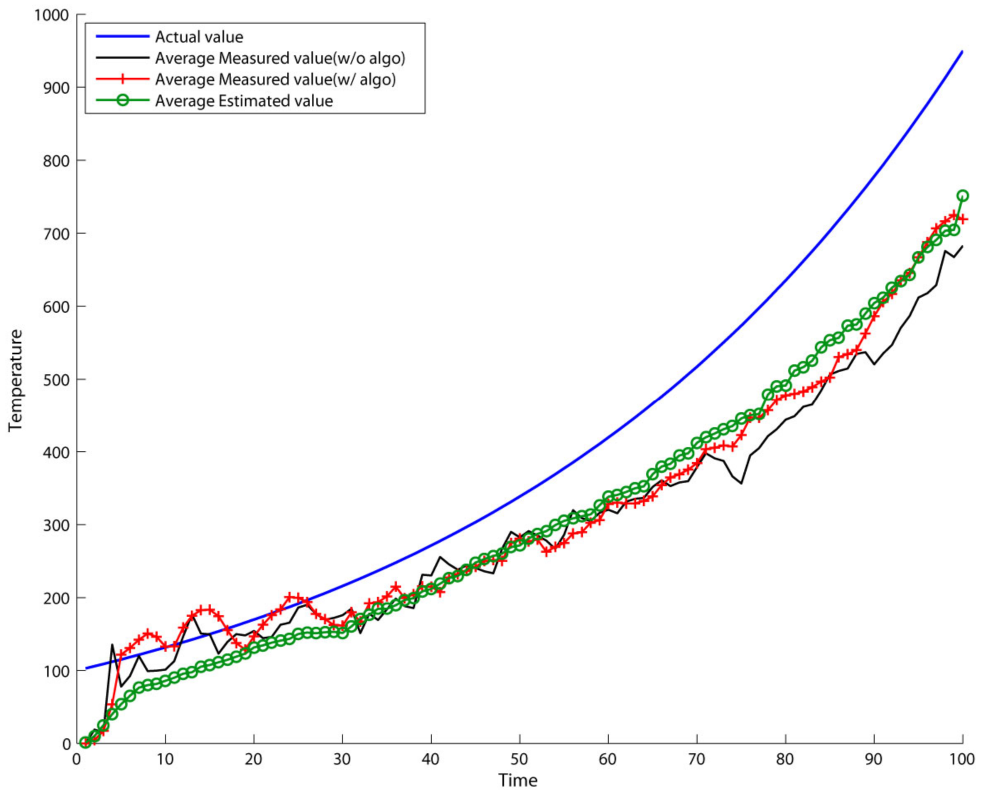


Figure 6.13 Heterogeneous swarm and algorithm with 10 nodes.

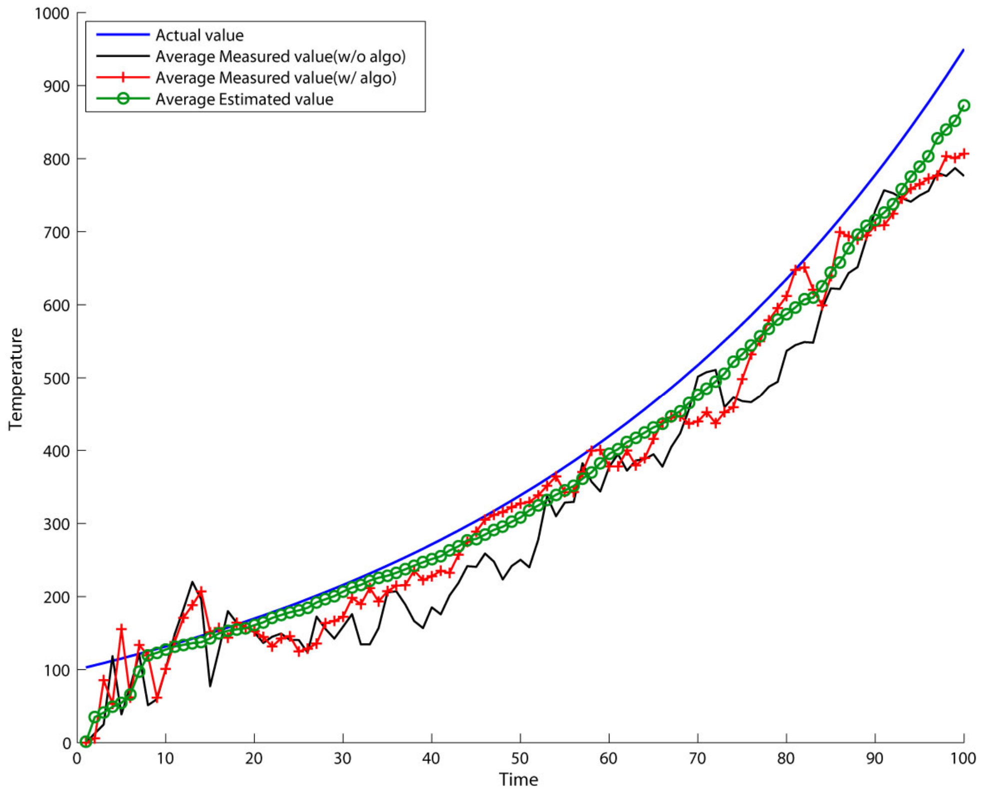


Figure 6.14 Heterogeneous swarm and algorithm with 30 nodes.

6.4 Heterogeneous swarm and homogeneous algorithm with Kalman filter

To perform an appropriate comparative analysis of the two proposed algorithms. The homogeneous algorithm is run in a heterogeneous swarm system. Thus, the homogeneous swarm does not account for the difference in the accuracy of the member nodes.

Figure 6.15 and figure 6.16 describe swarms with 10 and 30 nodes, respectively. All parameters are similar for respective models of the heterogeneous algorithm previously simulated. Comparing figure 6.13 And figure 6.15, it can be seen that beyond time=50 the heterogeneous algorithm provides better estimate. Also, in figure 6.14 and figure 6.16, a noticeable improvement is seen. The heterogeneous algorithm captures the true value faster and gives better result throughout. This is due to larger nodes and more number of accurate nodes. Thus, the better suited heterogeneous algorithm gives better results.

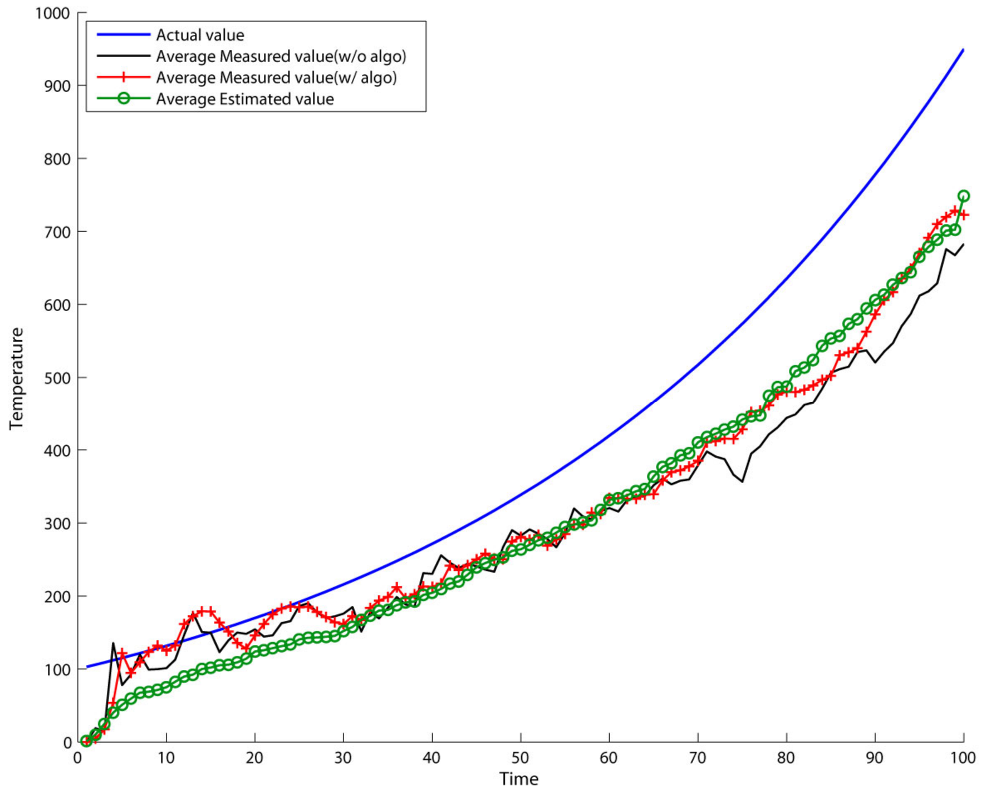


Figure 6.15 Heterogeneous swarm and homogeneous algorithm with 10 nodes.

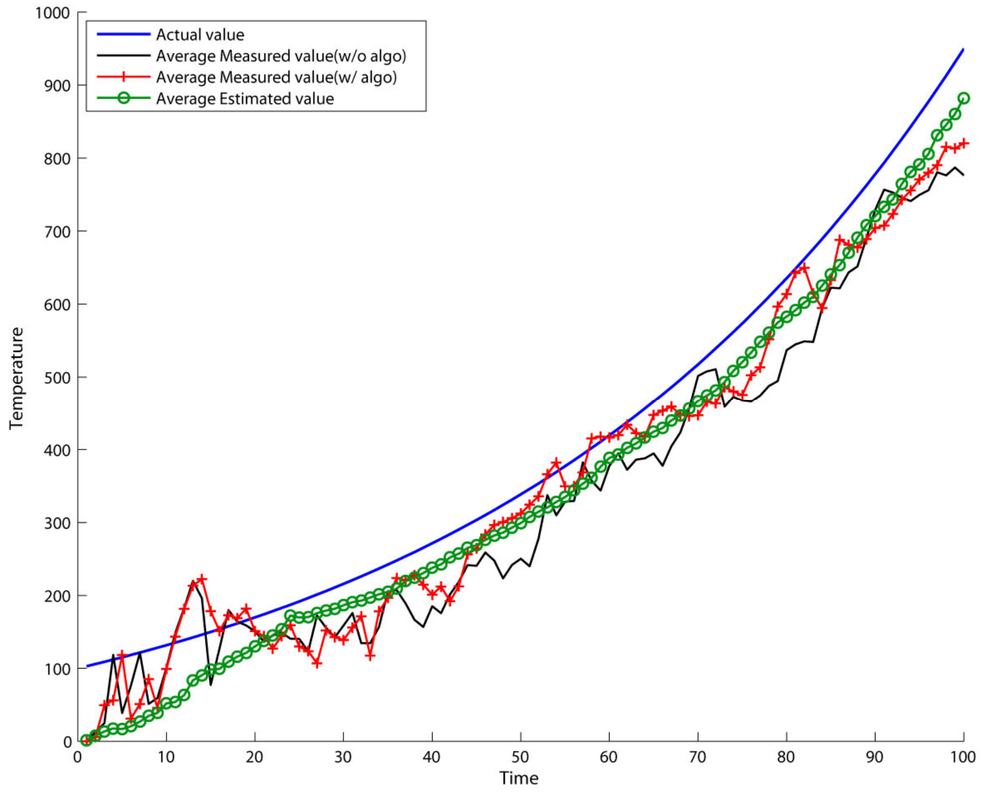


Figure 6.16 Heterogeneous swarm and homogeneous algorithm with 30 nodes.

7. Conclusion

In this study, the effectiveness of Kalman filter algorithm is analyzed in a homogeneous and heterogeneous swarm network in a noisy environment. Firstly, the behavior of a standard Kalman filter is examined. Then two sharing algorithms are proposed for both homogeneous and heterogeneous swarms. These swarms are designed to observe an underlying system with erroneous sensors. Thus, such swarms must utilize a noise filter to achieve a more accurate estimate. Each sharing algorithm uses two different types of sharing schemes to achieve a more accurate estimate in a distributed environment. Later, a comparative analysis is conducted between the two algorithms and as well as a standard moving average. It is clearly shown that given sufficient number of nodes the algorithms perform better than the moving average. Also, as expected, a heterogeneous system (with few more accurate nodes) provides better results than a homogeneous system. The effectiveness of the rank based heterogeneous algorithm is also examined. It is shown that in a heterogeneous environment, the heterogeneous algorithm performs better than the homogeneous algorithm. This is due to the fact that the heterogeneous algorithm takes into account the difference in capabilities of each node, thus, achieving higher accuracy and reduced time to capture the true value.

8. Future Work

There is a very broad scope of possible work in the future. General improvements in speed, response time and memory consumption can obviously be achieved. A dynamic approach to currently fixed parameters such as, $NSTAB$, $TDIFF$ and Rf, can be studies in the future. A dynamic switching between the homogeneous and heterogeneous nodes may be possible in cases where there are a large number of nodes of each type. Thus, if a large number of neighbours of a node are of the same type then homogeneous algorithm would be faster and more effective. The existing algorithm can easily be modified to work with EKF and UKF algorithms which would be effective in nonlinear systems. A comparative study of various filters may also be conducted. The proposed work has large applications in multiple fields. A modified and more specific system may be developed for different environments. For example, path traversal, searching, social network information sharing etc. can greatly benefit from the proposed work. Also, more in depth analysis would be helpful in identifying differences in behavior of homogeneous and heterogeneous swarms.

References

[1] Arulampalam, M. Sanjeev, Simon Maskell, Neil Gordon, and Tim Clapp. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking." Signal Processing, IEEE Transactions on 50, no. 2 (2002): 174-188.

[2] Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz. Swarm intelligence: from natural to artificial systems. Vol. 4. New York: Oxford university press, 1999.

[3] Brooks, Richard R., and Sundararaja S. Iyengar. Multi-sensor fusion: fundamentals and applications with software. Prentice-Hall, Inc., 1998.

[4] Buchsbaum, Daphna, Pablo Funes, Julien Budynek, Heiner Koppermann, and Eric Bonabeau. "Designing collective behavior in a group of humans using a real-time polling system and interactive evolution." In Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE, pp. 15-21. IEEE, 2005.

[5] Cai, N., and Y-S. Zhong. "Formation controllability of high-order linear time-invariant swarm systems." Control Theory & Applications, IET 4, no. 4 (2010): 646-654.

[6] Deming, R., L. Perlovsky, and R. Brockett. "Sensor fusion for swarms of unmanned aerial vehicles using modeling field theory." In Integration of Knowledge Intensive Multi-Agent Systems, 2005. International Conference on, pp. 122-127. IEEE, 2005.

[7] Doucet, Arnaud, Nando De Freitas, and Neil Gordon. Sequential Monte Carlo methods in practice. Vol. 1. New York: Springer, 2001.

[8] Ducatelle, Frederick, Gianni A. Di Caro, and Luca M. Gambardella. "Cooperative self-organization in a heterogeneous swarm robotic system." In Proceedings of the 12th annual conference on Genetic and evolutionary computation, pp. 87-94. ACM, 2010.

[9] Hall, David L., and James Llinas, eds. Multisensor data fusion. CRC press, 2001.

[10] Julier, Simon J., and Jeffrey K. Uhlmann. "New extension of the Kalman filter to nonlinear systems." In AeroSense'97, pp. 182-193. International Society for Optics and Photonics, 1997.

[11] Kalman, Rudolf Emil. "Contributions to the theory of optimal control." Bol. Soc. Mat. Mexicana 5, no. 2 (1960): 102-119.

[12] Kalman, Rudolph E., and Richard S. Bucy. "New results in linear filtering and prediction theory." Journal of Basic Engineering 83, no. 3 (1961): 95-108.

[13] Kalman, Rudolph Emil. "A new approach to linear filtering and prediction problems." Journal of basic Engineering 82, no. 1 (1960): 35-45.

[14] Karaboga, Dervis, and Bahriye Akay. "A survey: algorithms simulating bee swarm intelligence." Artificial Intelligence Review 31, no. 1-4 (2009): 61-85.

[15] Karaboga, Dervis. "An idea based on honey bee swarm for numerical optimization." Techn. Rep. TR06, Erciyes Univ. Press, Erciyes (2005).

[16] Kumar, Manish, Devendra P. Garg, and Randy A. Zachery. "A method for judicious fusion of inconsistent multiple sensor data." Sensors Journal, IEEE 7, no. 5 (2007): 723-733.

[17] Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." Communications of the ACM 21, no. 7 (1978): 558-565.

[18] Li, Ling, Alcherio Martinoli, and Yaser S. Abu-Mostafa. "Learning and measuring specialization in collaborative swarm systems." Adaptive Behavior 12, no. 3-4 (2004): 199-212.

[19] Li, Weihua, Sirish L. Shah, and Deyun Xiao. "Kalman filters in non-uniformly sampled multirate systems: for FDI and beyond." Automatica 44, no. 1 (2008): 199-208.

[20] Lo, Virginia, Dayi Zhou, Yuhong Liu, Chris GauthierDickey, and Jun Li. "Scalable supernode selection in peer-to-peer overlay networks." In Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on, pp. 18-25. IEEE, 2005.

[21] Magg, Sven, and René te Boekhorst. "Pattern formation in homogeneous and heterogeneous swarms: differences between versatile and specialized agents." In Artificial Life, 2007. ALIFE'07. IEEE Symposium on, pp. 311-316. IEEE, 2007.

[22] Maybeck, Peter S. Stochastic models, estimation, and control. Vol. 3. Access Online via Elsevier, 1982.

[23] Olfati-Saber, Reza. "Distributed Kalman filter with embedded consensus filters." In Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on, pp. 8179-8184. IEEE, 2005.

[24] Olfati-Saber, Reza. "Distributed Kalman filtering for sensor networks." In Decision and Control, 2007 46th IEEE Conference on, pp. 5492-5498. IEEE, 2007.

[25] Parunak, H. Van Dyke, and Sven A. Brueckner. "Engineering swarming systems." In Methodologies and Software Engineering for Agent Systems, pp. 341-376. Springer US, 2004.

[26] Roumeliotis, Stergios I., and George A. Bekey. "Collective localization: A distributed Kalman filter approach to localization of groups of mobile robots." In

Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, vol. 3, pp. 2958-2965. IEEE, 2000.

[27] Sorenson, Harold W. "Least-squares estimation: from Gauss to Kalman." Spectrum, IEEE 7, no. 7 (1970): 63-68.

[28] Tanenbaum, Andrew S., and Maarten Van Steen. Distributed systems. Vol. 2. Prentice Hall, 2002.

[29] Vatankhah, Ramin, Shahram Etemadi, Mohammad Honarvar, Aria Alasty, Mehrdad Boroushaki, and Gholamreza Vossoughi. "Online velocity optimization of robotic swarm flocking using particle swarm optimization (PSO) method." In Mechatronics and its Applications, 2009. ISMA'09. 6th International Symposium on, pp. 1-6. IEEE, 2009.

[30] Wiener, Norbert. Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications. Vol. 8. MIT press, 1964.

[31] Yang, Bo, Yunping Chen, Zunlian Zhao, and Qiye Han. "A master-slave particle swarm optimization algorithm for solving constrained optimization problems." In Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on, vol. 1, pp. 3208-3212. IEEE, 2006.

[32] Yang, Mao, Chengfeng Li, and Yantao Tian. "Flocking for Swarm Robot System: Distributed Coadaptive Control and Optimization." Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on. IEEE, 2009.

[33] Zhang, Hong-juan, and Hong-yun Ning. "Dynamic Parameters Ant Colony Algorithm with Particle Swarm Characteristic." In Natural Computation, 2009. ICNC'09. Fifth International Conference on, vol. 3, pp. 241-245. IEEE, 2009.

APPENDIX A: Code Snippets

Following are MATLAB code snippets for sharing methods.

ShareSimpleHo(1,2)

```
nd2.x_esti_post = nd1.x_esti_post;
nd2.P_esti_post = nd1.P_esti_post;
nd2.tlast = nd1.tlast;
for i=nd1.Nm:-1:nd1.yt_send_point
        nd2.y_measure_list(i) = nd1.y_measure_list(i);
        nd2.t_measure_list(i) = nd1.t_measure_list(i);
end
nd2.Nm = nd1.Nm;
nd2.yt_send_point = nd1.yt_send_point;
```

ShareComplexHo(1,2)

```
%2->1
nd1.x_esti_post = nd2.x_esti_post;
nd1.P_esti_post = nd2.P_esti_post;
nd1.tlast = nd2.tlast;
%1 process
x = nd1.timeIndexSearch(nd2.tlast);
cnt = 0;
for i=x:1:nd1.Nm
        nd1.kalmanIterate(i);
        cnt = cnt + 1;
end
for i=nd1.Nm-cnt:-1:nd1.yt_send_point
        nd1.y_measure_list(i) = nd2.y_measure_list(i+cnt);
        nd1.t_measure_list(i) = nd2.t_measure_list(i+cnt);
end
for i=nd1.Nm:-1:nd1.yt_send_point
        nd1.y_measure_list(i+cnt) = nd1.y_measure_list(i);
        nd1.t_measure_list(i+cnt) = nd1.t_measure_list(i);
end
nd1.Nm = nd2.Nm + cnt;
nd1.yt_send_point = nd1.yt_send_point + cnt;
%1->2
nd2.x_esti_post = nd1.x_esti_post;
nd2.P_esti_post = nd1.P_esti_post;
nd2.tlast = nd1.tlast;
for i=nd1.Nm:-1:nd1.yt_send_point
        nd2.y_measure_list(i) = nd1.y_measure_list(i);
        nd2.t_measure_list(i) = nd1.t_measure_list(i);
end
```

```
nd2.Nm = nd1.Nm;
nd2.yt_send_point=nd1.yt_send_point;
```

ShareSimpleHe(1,2)

```
nd2.x_esti_post = nd1.x_esti_post;
nd2.P_esti_post = nd1.P_esti_post;
nd2.tlast = nd1.tlast;
for i=nd1.Nm:-1:nd1.yt_send_point
        nd2.y_measure_list(i) = nd1.y_measure_list(i);
        nd2.t_measure_list(i) = nd1.t_measure_list(i);
end
nd2.Nm = nd1.Nm;
nd2.yt_send_point = nd1.yt_send_point;
nd2.avg_rank = nd1.avg_rank;
```

ShareComplexHe(1,2)

```
nd1.x_esti_post = nd2.x_esti_post;
nd1.P_esti_post = nd2.P_esti_post;
nd1.tlast = nd2.tlast;
%1 process
x = nd1.timeIndexSearch(nd2.tlast);
cnt = 0;
for i=x:1:nd1.Nm
        nd1.kalmanIterate(i);
        cnt = cnt + 1;
end
for i=nd1.Nm-cnt:-1:nd1.yt_send_point
        nd1.y_measure_list(i) = nd2.y_measure_list(i+cnt);
        nd1.t_measure_list(i) = nd2.t_measure_list(i+cnt);
end
for i=nd1.Nm:-1:nd1.yt_send_point;
        nd1.y_measure_list(i+cnt) = nd1.y_measure_list(i);
        nd1.t_measure_list(i+cnt) = nd1.t_measure_list(i);
end
nd1.Nm = nd2.Nm + cnt;
nd1.yt_send_point = nd1.yt_send_point + cnt;
nd1.avg_rank = (nd2.avg_rank * nd2.Nm + nd1.avg_rank * cnt) / (nd2.Nm + cnt);
%1->2
nd2.x_esti_post = nd1.x_esti_post;
nd2.P_esti_post = nd1.P_esti_post;
nd2.tlast = nd1.tlast;
for i=nd1.Nm:-1:nd1.yt_send_point
        nd2.y_measure_list(i) = nd1.y_measure_list(i);
        nd2.t_measure_list(i) = nd1.t_measure_list(i);
end
nd2.Nm = nd1.Nm;
```

```
nd2.yt_send_point = nd1.yt_send_point;
nd2.avg_rank = nd1.avg_rank;
```