Fall 11-2016

Investigation of Communication Constraints in Distributed Multi-Agent Systems

Zhaoyang Fu
Embry-Riddle Aeronautical University

INVESTIGATION OF COMMUNICATION CONSTRAINTS IN DISTRIBUTED MULTI-AGENT SYSTEMS

A Thesis

Submitted to the Faculty

of Embry-Riddle Aeronautical University

by

Zhaoyang Fu

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

November 2016

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Investigation of Communication Constraints in Distributed Multi-agent Systems

by

Zhaoyang Fu

A Thesis prepared under the direction of the candidate's committee chairman, Dr. Tianyu Yang, Department of Electrical, Computer, Software & Systems Engineering, and has been approved by the members of the thesis committee. It was submitted to the School of Graduate Studies and Research and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering.

THESIS COMMITTEE

[illegible]
Chairman, Dr. Tianyu Yang

[illegible]
Member, Dr. Ilteris Demirkiran

[illegible]
Member, Dr. Shuo Pang

[illegible] Department Chair, Dr. Timothy A. Wilson or Graduate Program Coordinator, Dr. Jianhua Liu	11/28/2016 Date
[illegible] Dean of College of Engineering, Dr. Maj Mirmirani	11/29/16 Date
[illegible] Vice Chancellor for Academics, Dr. Christopher D. Grant	11/29/16 Date

Acknowledgments

I would like to express the sincerest appreciation to my committee chair Dr. Tianyu Yang, for his patience, time, advice, help, mentoring and clarity. His supervision, wisdom and aid in direction greatly contributed to making this thesis possible, and for that I am deeply grateful.

I would also like to thank my committee members, Dr. Ilteris Demirkiran and Dr. Shuo Pang. They have all always been available to help me solve all queries in my research and writing work. They steered me in the right direction to ensure successful completion.

In addition, I would like to thank the support from the U.S. Air Force Research Laboratory/Information Directorate for my research.

Last, but not least, I would like to thank my family for their support, love, and inspiration throughout my life and academic career.

ABSTRACT

Researcher: Zhaoyang Fu

Title: Investigation of Communication Constraints in Distributed Multi-agent Systems

Institution: Embry-Riddle Aeronautical University

Degree: Master of Science in Electrical and Computer Engineering

Year: 2016

Based on a simple flocking model with collision avoidance, a set of investigations of multi-agent system communication constraints have been conducted, including distributed estimation of global features, the influence of jamming, and communication performance optimization. In flocking control, it is necessary to achieve a common velocity among agents and maintain a safe distance between neighboring agents. The local information among agents is exchanged in a distributed fashion to help achieve velocity consensus. A distributed estimation algorithm was recently proposed to estimate the group's global features based on achieving consensus among agents' local estimations of such global features. To reduce the communication load, the exchange of local estimations among agents occurs at discrete time instants defined by an event-triggering mechanism. To confirm the effectiveness of the new distributed estimation algorithm, we simulated the algorithm while adopting a simple flocking control technique with collision avoidance. In addition, the effect of jamming on flocking control and the distributed algorithm is studied through computer simulations. Finally, to better exploit the communication channel among agents, we study a recently proposed formation control multi-agent algorithm, which optimizes the inter-agent distance in order to achieve optimum inter-agent communication performance. The study is also conducted through computer simulations, which confirms the effectiveness of the algorithm.

Table of Contents

Table of Figures

List of Key Symbols

a_{ij}: agents j's influence on agent i in the simple flocking model

d_0: minimum distance in repelling force in the simple flocking model

c_{ij}: communication matrix index in the distributed estimation algorithm

μ_i: agent i's contribution to the network moment in the distributed estimation algorithm

$\hat{\mu}_i$: agent i's estimation of the network moment in the distributed estimation algorithm

c_0 and α: constants used in defining the threshold in event-triggering time sequence in the distributed estimation algorithm

u_i: agent i's control input in the communication-aware formation control

α: antenna characteristics parameter used in the communication-aware formation control

δ: required application data rate in the communication-aware formation control

v: path loss exponent used in in the communication-aware formation control

r_0: a reference distance for the antenna near-field used in the communication-aware formation control

r: the distance between transmitter and receiver used in the communication-aware formation control

P_T: the reception probability threshold defined in the communication-aware formation control

$\psi(r_{ij})$: the artificial potential function in the communication-aware formation control

$\phi(r_{ij})$: the communication performance indicator defined in the communication-aware formation control

Chapter 1: Introduction

1.1 Background and Literature Search

In a natural world, a swarm consists of many similar agents, such as: a flock of birds or a group of fish [1]-[3]. The interactions among the agents can be simple or more complex, and can occur between neighbors in space or in an underlying network. The main feature of swarm is that an individual unit's action is dominated by the influence of "others". In recent years, researchers have been attempting to apply the principle of natural swarms to bio-inspired manmade systems, e.g., a group of robots, unmanned aerial vehicles (UAVs) and autonomous underwater vehicles (AUVs), or even mobile sensors [13][23]. A bio-inspired system can be constructed in such a way that the control of the entire group can be achieved through controlling a small number of agents.

Figure 1.1. A school of fish (source: https://pando.com/2012/12/04/users-swarm-to-summly-one-month-post-redesign/)

Figure 1.2. A swarm of birds (source: http://www.martinemaes.nl/geef-ruimte/)

For example, in the Couzin's model of biological swarms and the Reynold model of synthetic agents, agents react to neighbors within three different zones: repulsion, orientation, and attraction [6]. An agent is repelled from neighbors within its repulsion zone of radius R_r, orients its heading with neighbors in its orientation zone of radius R_0, and is attracted to neighbors outside of its orientation zone. The angular velocity ω_i is determined by summing the desired direction vectors resulting from the repulsion, orientation, and attraction rules.

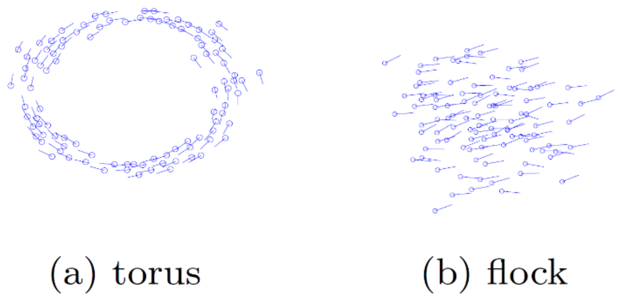

Figure1. 3. Two types of flocking formation [14].

In nature, flocks can be considered self-organized networks of mobile agents, which are able to coordinate the group behaviors. Neighbor-based approaches are widely applied in multi-agent coordination, inspired originally by the aggregations of groups of individual agents in nature. Multi-agent systems typically need distributed estimations and control laws due to the constraints on actuation, communication and measurement.

Consensus problem is a very important part in the multi-agent research history and it forms the foundation of the field of distributed computing [19]. The study of multi-agent consensus problems originated from the management science and statistics in 1960s [10]. The ideas of statistical consensus theory proposed by

DeGroot appeared two decades later during investigation of information with uncertainty obtained from multiple sensors [33] [34].

Distributed computation of networks originated from systems and control theory starting with the pioneering work of Borkar, Varaiya [12], Tsitsiklis [27], Bertsekas, and Athans [28] on agreement problem for distributed decision making systems, and parallel computing [4].

In 1986, Reynolds introduced three rules for the creation of the first computer animation of flocking [17], which are:

1) Flock Centering: attempt to stay close to nearby neighboring agents.

2) Collision Avoidance: avoid collisions with nearby neighboring agents.

3) Velocity Matching: attempt to match velocity with nearby neighboring agents.

To further consider multi-agent systems with complicated dynamics, the Reynolds' flocking rules were embedded into several control methods and strategies, which are behavior-based method, leader-follower method, virtual structure method.

Since then, more and more physicists made much effort on flocking studies. Among the first groups of physicists who studied the theoretical perspective were Vicsek et al. (1995) [18], Toner and Tu (1998)[29], Shimoyama et al. (1996) [25], and Levine. The work of Vicsek was mainly focused on emergency behaviors of alignment in self-driven particle systems. Toner and Tu used a continuum mechanics approach. Levine created rotating swarms using a particle-based model with all-to-all interactions. Also, Mogilner and Eldstein-Keshet (1999) [22] and Topaz and Bertozzi Helbing (2000) proposed other continuum models of swarms [24].

The study of distributed control of multiple agents was perhaps first motivated by the work in distributed computing [19], management science [10], and statistical physics [25]. In the control systems research community, the so-called agreement problem was studied for distributed decision-making applications [27]. Distributed estimation by observation for multi-agent system is an important topic in the study of multi-agent networks, with wide variety of applications, especially in sensor networks and robot systems. So far, there are many results obtained on distributed observer design and measurement-based dynamic multi-agent control design. Fax and Murray (2004) reported some results concerning distributed dynamic feedback of special multi-agent networks, and Hong, Hu, and Gao (2006) also proposed an algorithm for distributed estimation of the active leader's unmeasurable state variables.

Besides, communication jamming is an important concern in various military and commercial applications of multi-agent systems. Jamming can be a malicious attack whose objective is to disrupt the communication of the victim network intentionally, causing interference or collision at the receiver side. In some applications, the jammer may be intentional and aims at disrupting the inter-agent communication [30], and in other applications the jamming may be unintentional and caused by communication interferences from other geographically collocated systems. There are generally four types of intentional jamming strategies: constant jammer, deceptive jammer, random jammer and reactive jammer [21].

1.2 Goals and Summary of the Current Research Work

The goal of this thesis is to investigate the communication constraints in distributed multi-agent system. We adopt a simple flocking model with collision avoidance, and apply a recently proposed distributed estimation algorithm to this flocking model, which estimates certain global features of the multi-agent systems through consensus of agents' local estimations. We also study effects of jamming on a multi-agent flocking model with distributed estimation of global features. In addition, the optimization of inter-agent communication link is studied through guiding the multi-agent system to achieve the optimum inter-agent distance for best communication performance.

1.3 Thesis Outline

Chapter 2 briefly describes the adopted simple flocking model with a straightforward collision avoidance mechanism, and presents the computer simulation results to illustrate the effectiveness of the model.

Chapter 3 summarizes the recently proposed algorithm for distributed estimation of global features, and presents the computer simulation results to validate the effectiveness of the algorithm.

Chapter 4 introduces a simple communication jamming model, and presents the computer simulation results to study the impact of jamming.

Chapter 5 studies a communication-aware formation control approach with the objective of optimizing the inter-agent communication performance, and presents the computer simulation results to confirm the validity of the method.

Chapter 6 concludes the thesis and briefly mentions future works.

Chapter 2: A Simple Flocking Model with Collision Avoidance

In this chapter, we study a simple flocking model with a straightforward collision avoidance mechanism. The collision avoidance is realized through a repelling force between agents moderated by an alignment measure.

2.1 Methodology

The collective motion of bird flocks, fish schools, or colonies of bacteria [5, 7] inspired many researchers' attempts to develop models for mobile autonomous agents [1,16]. The most fascinating fact about these natural phenomena is that, local behaviors of individual agents often lead to emergent global behaviors through only intermittent interactions among neighboring agents. The absence of centralized control offers significant potential benefits for man-made multi-agent systems [13][18][26], such as unmanned aerial systems (UAS) and autonomous underwater vehicles (AUV).

The achievement of consensus in velocity among all agents is the goal of many multi-agent system models, which are sometimes referred to as flocking models. The interaction between agents in such models often includes repulsion, attraction, and orientation [15]. We adopt a simple Laplacian-based flocking model with collision avoidance, and implement it through computer simulations in three-dimension. The model guarantees the formation of a cohesive group through a convergence process that is free of agent collision. The collision avoidance is achieved through a repelling force moderated by a measure of group alignment [8].

We present the agent dynamics and the multi-agent flocking model employed in our simulations. Assume that in a 3D space, at time $t = 0$, k agents are randomly distributed in their initial positions and have the same absolute velocity. The agent dynamics are:

$$\begin{aligned} \dot{x}_i &= v_i cos\theta cos\psi \\ \dot{y}_i &= v_i cos\theta sin\psi \\ \dot{z}_i &= v_i sin\theta \end{aligned} \tag{1}$$

where the driving velocity is v_i. We denote $\dot{x}_i$, $\dot{y}_i$and $\dot{z}_i$ as the three axes velocity components of the i^{th} agent. The inclination $\theta_i \in [0,\ \pi]$, the azimuth $\psi_i \in [0,2\pi)$. In this case, $(\dot{x}_i\ , \dot{y}_i, \dot{z}_i)\ \in\ \mathbb{R}^3$, and $i \in 1, \dots ,\ k$.

To achieve consensus in velocity of all agents in the group, each agent adjusts its velocity to match that of its neighbors, i.e., we seek rules for all agents to follow with the objective to achieve an equilibrium condition, in which all agents have the same values of velocity and acceleration. The velocities and positions of all agents are updated at each time step. According to the popular Laplacian-based model [9], every agent adjusts its velocity by adding to it a weighted average of the differences of its velocity with those of the other agents. That is, at time t for agent i,

$$\begin{aligned} x_i(t+h) &= x_i(t) + hv_i(t) \\ v_i(t+h) &= h\sum_{j=1}^{N} a_{ij}(v_j(t) - v_i(t)) + v_i(t) \end{aligned} \tag{2}$$

where $h > 0$ is the time step. $x_i(t)$ and $v_i(t)$denote the position and velocity of agent i at time t. The weights a_{ij} qualify the degree the agents influence each other, which depends on the distance $d(x_i, x_j)$ between x_i and x_j. To make this dependence non-increasing, the celebrated Vicsek's model chooses

$$a_{ij} = \begin{cases} 1, & if \ \|x_i - x_j\| \leq R \\ 0, & otherwise. \end{cases} \tag{3}$$

This equation can be rewritten in

$$a_{ij}(x) = \frac{H}{(1 + ||x_i - x_j||^2)^{\beta}} \tag{4}$$

where $H > 0$ and $\beta > 0$ are system parameters. The convergence of the Vicsek's model has been rigorously proven [31].

In order to achieve collision avoidance, the agents have the additional task of maintaining the safe distance from others. A common way to do so is to add a repelling force that is in effect whenever a pair of agents get close to each other, and the strength of such force should increase as the pair of agents get closer. A repelling force should satisfy two conditions, which are shown below.

$$\begin{aligned} f(1)&: \int_{d_0}^{d_{0+1}} f(r)dr = \infty \\ f(2)&: \int_{d_{0+1}}^{\infty} f(r)dr < \infty \end{aligned} \tag{5}$$

The repelling force function should take effect within a certain sensor range d_0. In our simulations, the function is taken as

$$f(r) = (r - d_0)^{-\theta} \tag{6}$$

To properly incorporate the repelling force function in the model, according to [9], the alignment measure is defined as:

$$\Lambda(v) = \left(\frac{1}{k}\sum_{i>j}\|v_i - v_j\|^2\right)^{\frac{1}{2}} \tag{7}$$

It is obvious that when the agents' velocities reach a consensus, the alignment measure becomes zero. We use the alignment measure to moderate the repelling force.

Now, the model with collision avoidance can be presented as the following system of differential equations [38]:

$$x_i(t+h) = x_i(t) + hv_i(t)$$

$$v_i(t+h) = h\sum_{j=1}^{N} a_{ij}(v_j(t) - v_i(t)) + h\Lambda(v)\sum_{j\neq i} f(\|x_i - x_j\|^2)(x_i - x_j) + v_i(t)$$

(8)

2.2 Simulation Results

In our MATLAB simulations, we implemented the flocking model described in the previous section. The simulation is set up for a flock with $k = 25$ agents, and the agents start with randomly generated initial positions and velocities. Other parameters are set as $R = 40$, $d = 10$, h=0.1, and total simulation times t=100. If the distance between two agents is less than R, according to the weight calculation equation, we assume the index $a_{ij}(t)$=1. The results are shown in Figures 2.1-2.2. It is obvious that after some time all agents' velocities achieve consensus, which resulted in stable group formation.

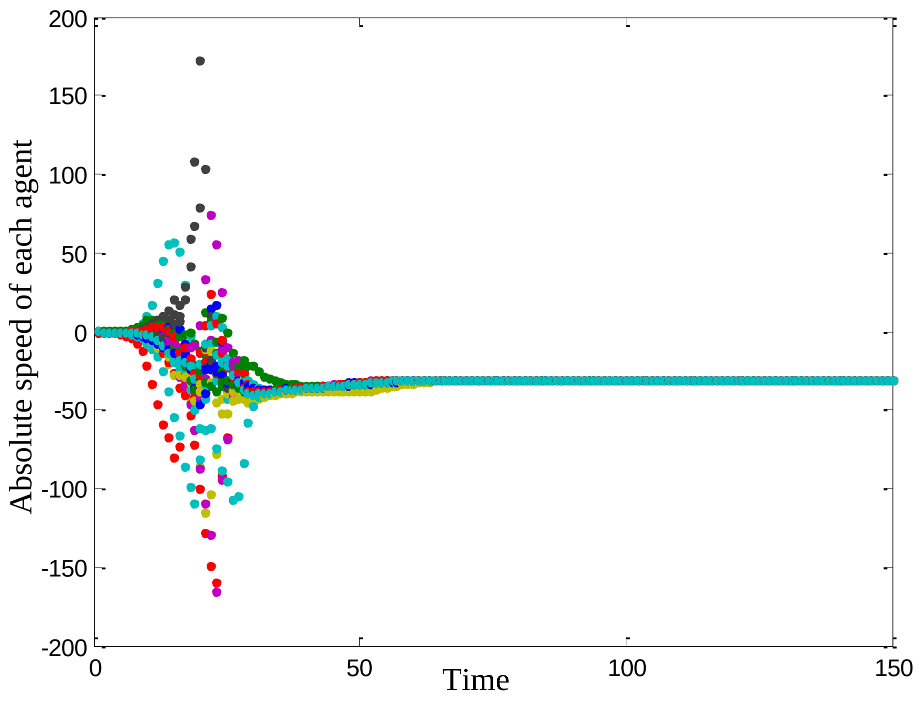

Figure 2.1. Speed with respect to time ($d = 1$)

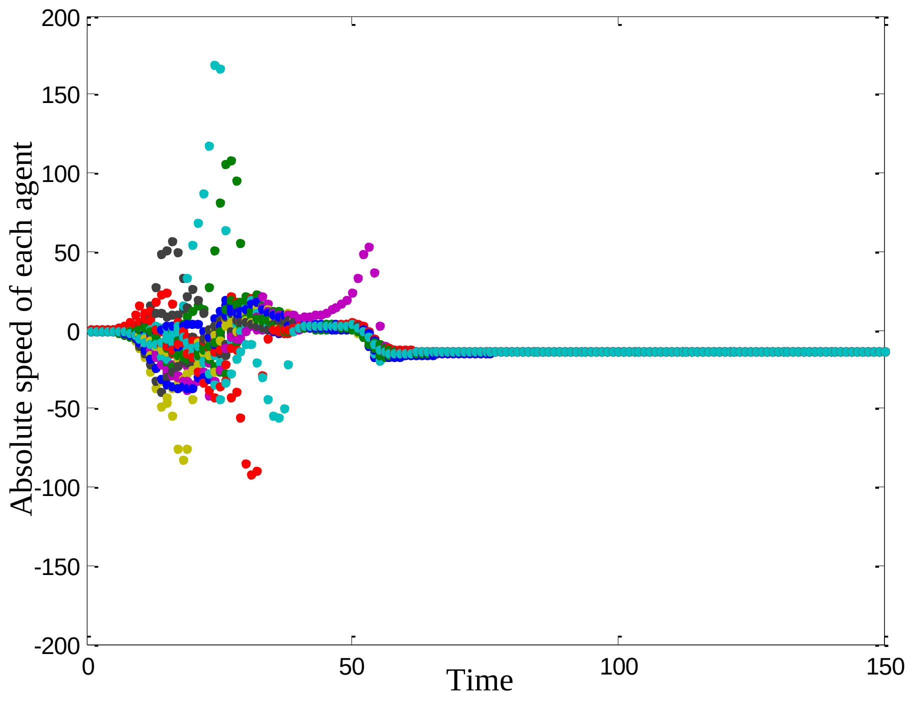

Figure 2.2. Speed with respect to time ($d = 10$)

Chapter 3: Event-Triggered Distributed Estimation Algorithm for Global Features in Multi-agent Systems

In this chapter, we investigate a recently proposed distributed estimation algorithm, while adopting the previous chapter's simple flocking model. The purpose of the estimation algorithm is to estimate global features in a distributed fashion, i.e., to obtain the global feature through consensus of individual agents' local estimations.

3.1 Methodology

To estimate global features of multi-agent systems (such as centroid, polarization, or momentum) in a distributed fashion, each agent in the system maintains a local estimation of the global feature, and updates the estimation based on information exchange with its neighbors [32][35]. When all agents' estimations converge, the true value of the global feature is obtained.

The information exchange among the swarm of agents can be expressed by the communication/sensing matrix

$$C(t) = \begin{bmatrix} 0 & c_{12}(t) & \cdots & c_{1N}(t) \\ c_{21}(t) & 0 & \cdots & c_{2N}(t) \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1}(t) & c_{N2}(t) & \cdots & 0 \end{bmatrix} \tag{9}$$

where $c_{ij}(t) > 0$ indicates that agent i can receive velocity and position information from agent j. Otherwise, $c_{ij}(t)$ = 0, which means there is no communication between agent i and agent j. We assume that communication network C is time-invariant, bidirectional, and connected. In practical applications, the communication matrix is determined by the agents' positions, signal interference levels, etc.

For agent i, we denote its estimate for network moment:

$$\hat{\mu}_i(t) = \frac{1}{N}\sum_{i=1}^{N}\int_0^t \mu_i(\tau)d\tau \tag{10}$$

The distributed estimation algorithm is expressed in the form of:

$$\dot{\hat{\mu}}_i(t) = \sum_{j\in\mathcal{N}_i} c_{ij}(t)\left(\hat{\mu}_j(t) - \hat{\mu}_i(t)\right) + \mu_i(t) \tag{11}$$

where $\mathcal{N}_i \triangleq \{j|c_{ij} > 0\}$ denotes the group of agents within the communication sensor range of agent i. In other words, agent i can receive the estimate $\hat{\mu}_j(t)$ of any agent $j \in \mathcal{N}_i$ and use this value to update its estimate $\hat{\mu}_i(t)$. The convergence of all agents' estimations is rigorously proven in [40].

In order to implement the estimation algorithm, we define the event-triggering time sequence for agents as $t_0, t_1, \ldots$ For $t \in [t_k, t_{k+1})$, we let

$$\hat{\mu}_i(t) = \hat{\mu}_i(t_k) \tag{12}$$

And the distributed estimation algorithm is given by

$$\dot{\hat{\mu}}_i(t) = \sum_{j\in\mathcal{N}_i} c_{ij}(t)\left(\hat{\mu}_j(t_k) - \hat{\mu}_i(t_k)\right) + \mu_i(t),\ t \in [t_k, t_{k+1}), \tag{13}$$

Define the measurement error as $\varepsilon = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N]^T$, where

$$\varepsilon_i(t) = \hat{\mu}_i(t_k)\text{-}\hat{\mu}_i(t) \tag{14}$$

When the summation of group measure error

$\varepsilon(t)$=$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\ \varepsilon_i(t)}$ fails to meet the condition (for some positive constants c_0 and α)

$$\|\varepsilon(t)\| \leq c_0 e^{-\alpha t}\ ,\ t \in [t_k, t_{k+1}) \tag{15}$$

the event-triggering time sequence $\{t_k\}$ will be updated. Therefore, this condition is used to decide when to request the transmission of $\hat{\mu}_j(t_k)$ and $\hat{\mu}_j(t_k)$. After transmission, the measurement error $\varepsilon_i(t_k)$ is automatically reset to zero.

3.2 Simulation Results

Based on the simulations described in the previous chapter, the distributed estimation algorithm was incorporated to estimate the location of group centroid, with k$= 22$, $c = 5$, $\alpha = 1$. If the distance between two agents is within the communication range, we assume the corresponding $c_{ij}(t)$=0.1. Figures 3.1-3.3 illustrate the convergence of all agents' estimations of the group centroid's position. Also, all agents' estimation errors decrease to zero, as shown in Figures 3.4-3.6.

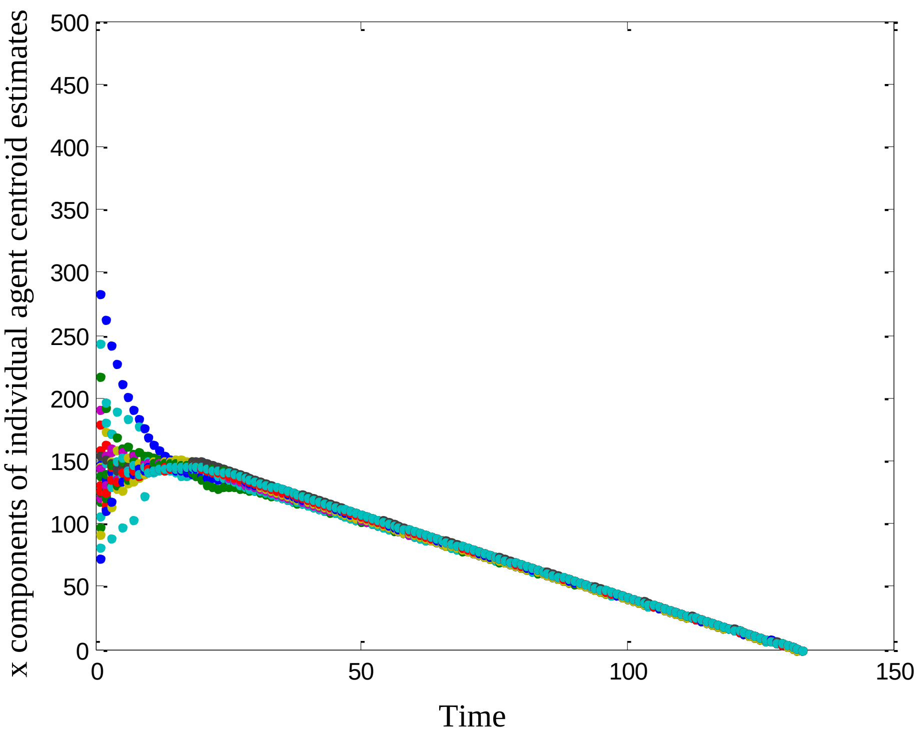

Figure 3.1. Estimates of $\frac{1}{N}\sum_{i=1}^{N} x_i(t)$ with respect to time

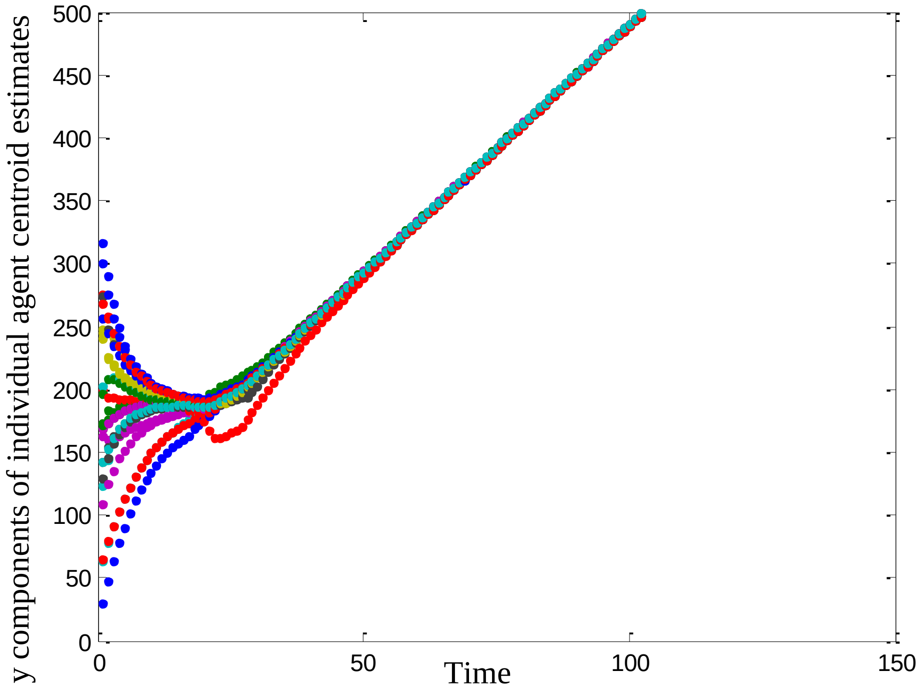

Figure 3.2. Estimates of $\frac{1}{N}\sum_{i=1}^{N} y_i(t)$ with respect to time

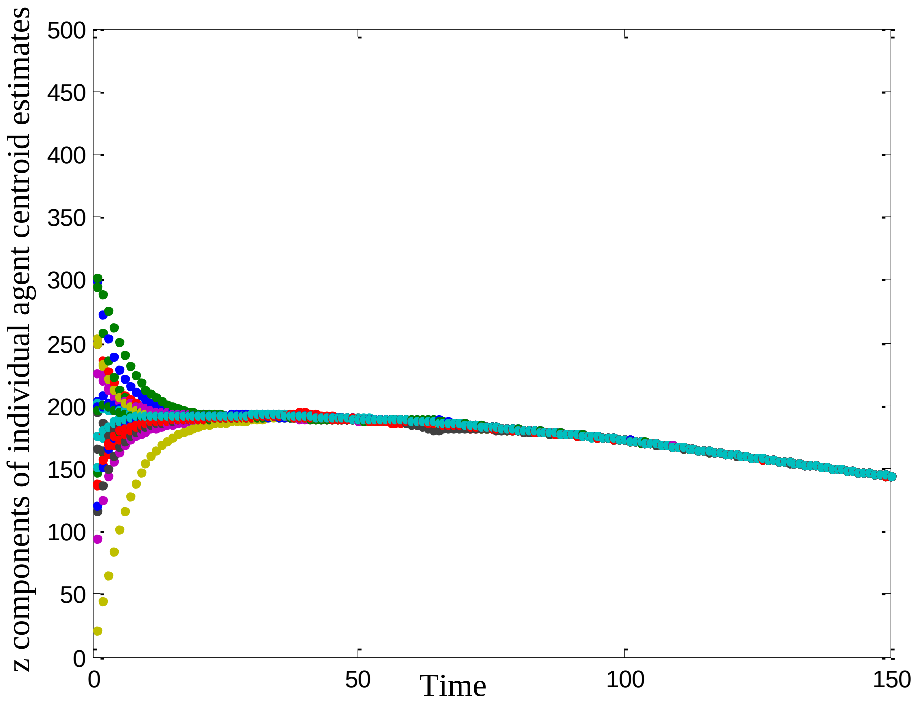

Figure 3.3. Estimates of $\frac{1}{N}\sum_{i=1}^{N} z_i(t)$ with respect to time

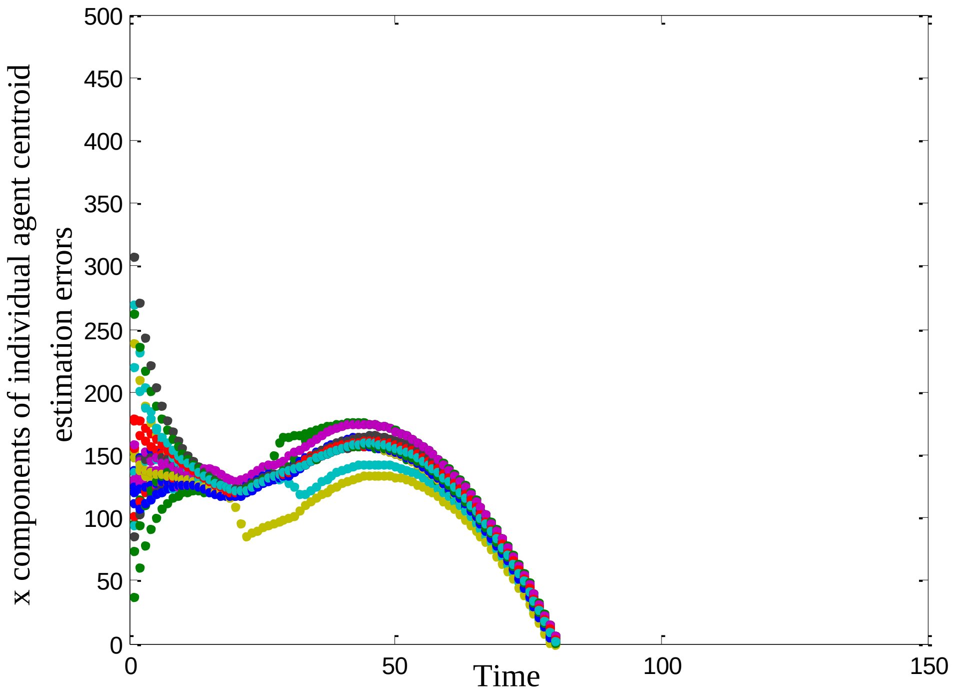

Figure 3.4. Estimation errors for $\frac{1}{N}\sum_{i=1}^{N} x_i(t)$ with respect to time

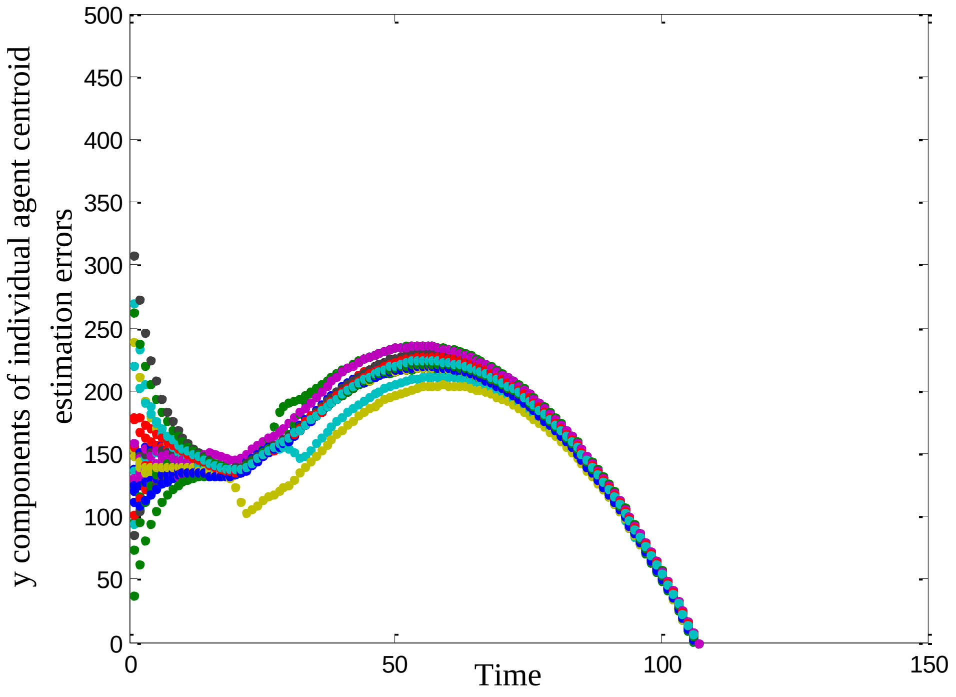

Figure 3.5. Estimation errors for $\frac{1}{N}\sum_{i=1}^{N} y_i(t)$ with respect to time

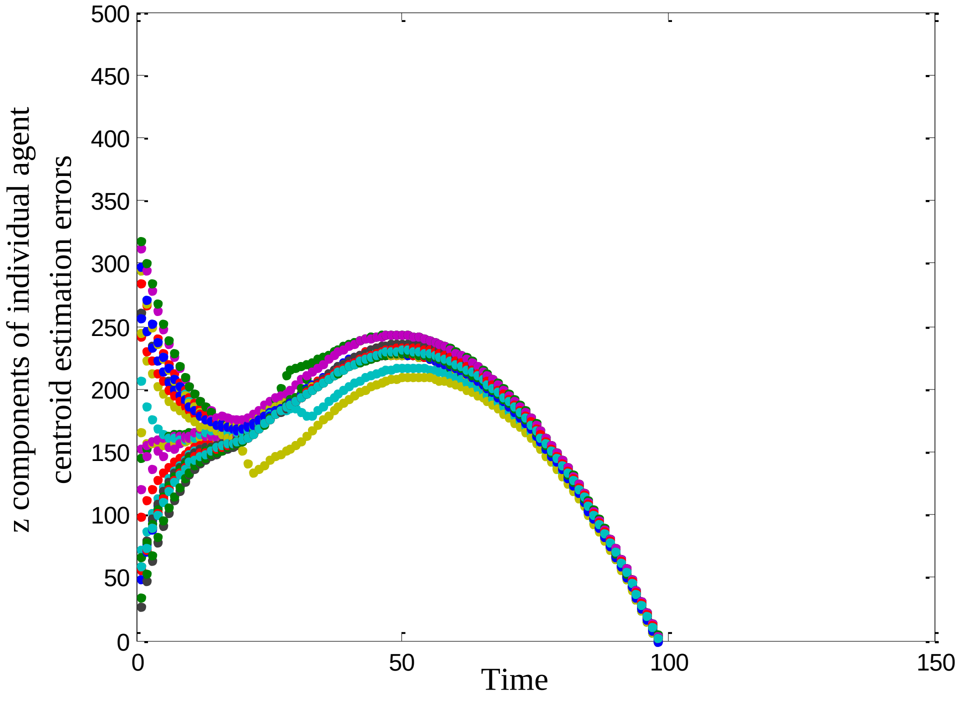

Figure 3.6. Estimation errors for $\frac{1}{N}\sum_{i=1}^{N} z_i(t)$ with respect to time

Chapter 4: Effects of Jamming on the Multi-agent Flocking Model with Distributed Estimation of Global Features

In this chapter, we study the effect of communication jamming on the achievement of multi-agent velocity and estimation consensuses.

4.1 Methodology

In our study of the effects of jamming, the simple flocking model with collision avoidance (as described in Chap. 2) and the recently proposed algorithm for distributed estimation of global features (as described in Chap. 3) are adopted. The objective of our research is to study the effects of jamming on the achievement of velocity and estimation consensuses.

Jamming normally refers to the transmission of radio signals by an adversary that disrupts communications through decreasing the signal-to-noise ratio. Unintentional jamming may also arise if a second radio transmission is initiated (without first checking the frequency band to be occupied) on a band currently used by a licensed user.

In military applications, a communication denied environment is often encountered, where jamming causes agents within certain area not able to communicate with neighboring agents. For agents in the jamming area, we assume they keep the same velocity until they flee from the jamming area, at which point they start to update their velocities again according to (8).

As shown in Figure 4.1, assume $J(x_j, y_j, z_j)$ denotes the location of a jammer, and the jamming area is defined in the 3D space as a sphere centered at $J(x_j, y_j, z_j)$ with radius R.

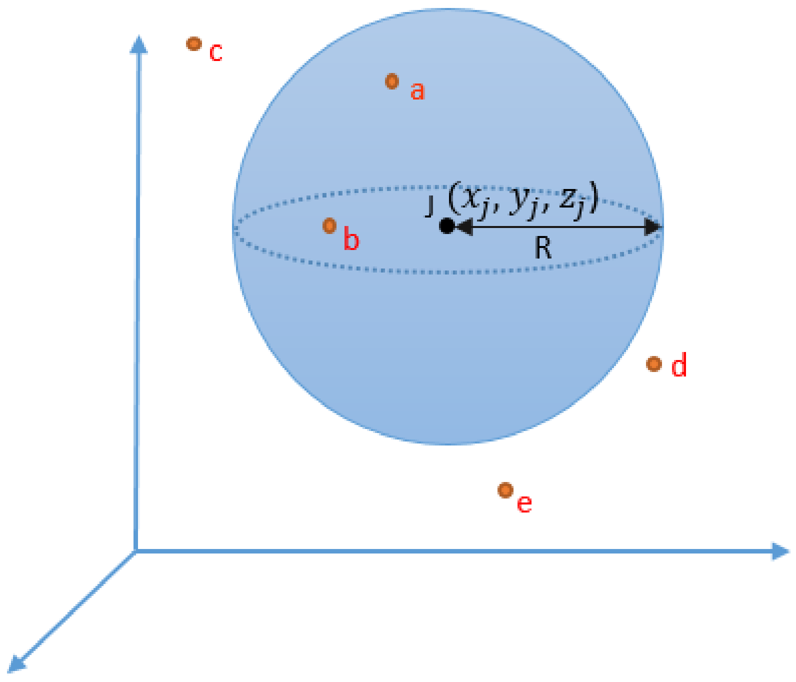

Figure 4.1. A jamming model of the multi-agent system

Assume the i^{th} agent's position is (x_i, y_i, z_i). The distance from the agent i to jamming center is:

$$Distance(i) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \qquad (16)$$

If $Distance(i) < R$, agent i is being jammed and cannot communicate with its neighbors. Otherwise, its communication with other agents is not affected [11].

In the example shown in Figure 4.1, the initial positions of five agents a, b, c, d and e are randomly distributed in the 3D space. The agents within the jamming area (agents a and b) are denied communication, so they keep their starting velocity until they leave the jamming area. Agents c, d and e are out of the jamming area, so their communication is not jammed initially, and they exchange position and velocity information with their neighbors based on (4), as long as the neighbors are within the communication range.

4.2 Simulation Results

In our MATLAB simulations, the consensus behavior of the flocking model and the distributed estimation algorithm are analyzed. The flocking group is considered to have reached consensus when the velocities of all agents converge to the same value, i.e., the alignment measure reaches approximately zero. The jamming center is set at the origin. Initially, all agents were randomly distributed over the range of [-25, 25] along all three axes. Simulations were run for both the simple flocking model and the distributed estimation algorithm. We present four sets of experimental results. In each set of simulations, the results were averaged over 100 Monte Carlo simulation runs. In the flocking model, we assume a_{ij}=1, time step $h = 0.5,$ and the number of agents is 25.

First, we focus on the velocity convergence time when the jamming range is varied from 5 to 50, incremented by 5 at a time. The communication range is set to 30. The result is shown in Figure 4.2, where it can be seen that as the jamming radius increases, the average convergence time increases roughly linearly from 10 to 85 time units.

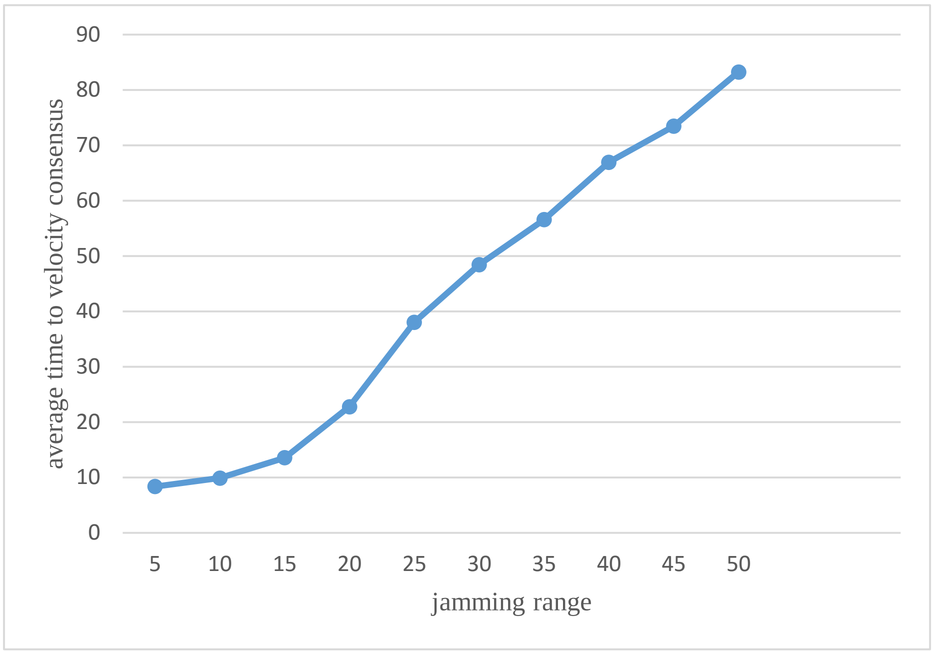

Figure 4.2. Average time to velocity consensus with respect to jamming range

Second, we study the effect of jamming on the delay of velocity convergence time with varied communication ranges. In our simulations, the communication range was increased from 5 to 50, with a step size of 5. By obtaining the convergence times with and without jamming, we calculate the delay of convergence time as a function of communication range. The result is shown in Figure 4.3, from which it is seen that the communication range has no significant effect on the delay of consensus, which remains approximately a constant of 40 time units.

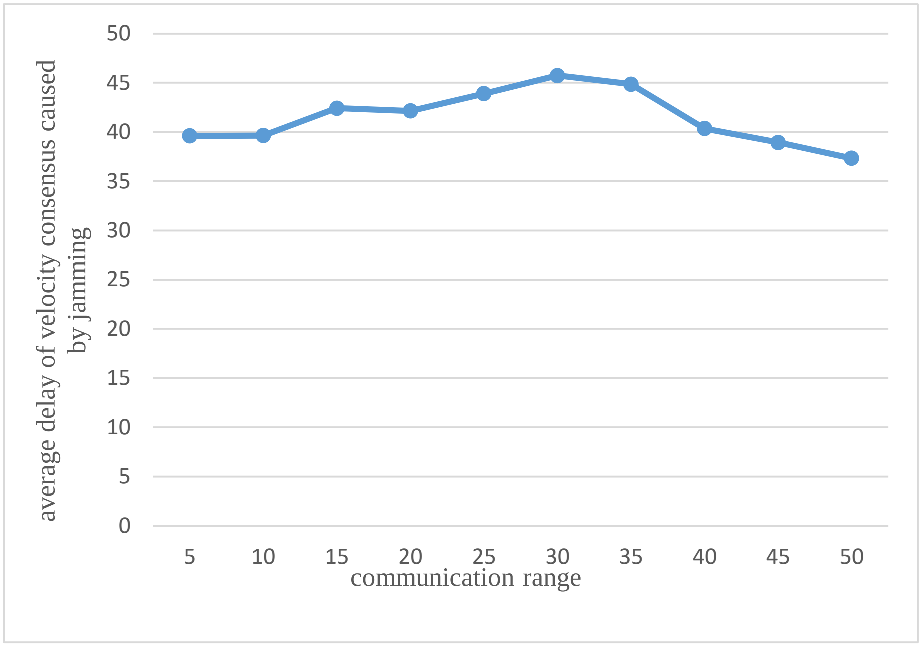

Figure 4.3. Average delay of velocity consensus caused by jamming with respect to communication range

Next, we simulated the newly proposed distributed estimation algorithm to estimate the location of group centroid. We assume $c_0 = 5$, $\alpha = 1$, and c_{ij}= 0.1 when the distance between two agents is less than the communication range.

In the third set of simulations, we study the agents' estimation convergence time with varied jamming ranges, and the result is shown in Figure 4.4. It is obvious that, larger jamming ranges result in longer time to reach consensus in agents' local estimates, and the relationship is approximately linear.

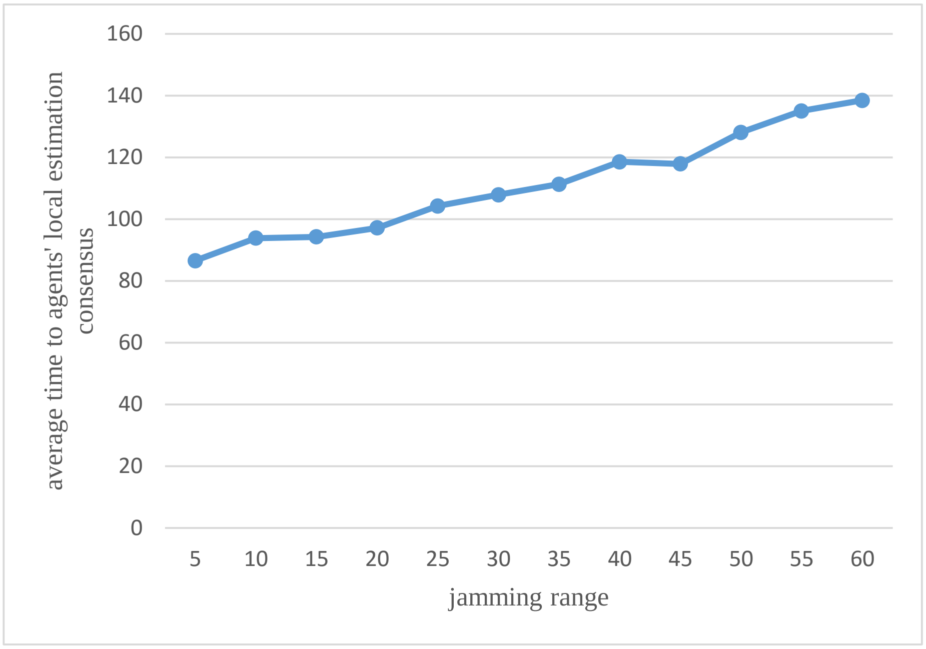

Figure 4.4. Average time to agents’ local estimation consensus with respect to jamming range

The last set of simulations, we present the average delay of agents’ estimation consensus as a function of the communication range, which increases from 5 to 50 with a step size of 5. The result is shown in Figure 4.5, which shows the delay in reaching agents’ estimation consensus is not significantly affected by different communication ranges. The delay stays at approximately a constant of 42 time units.

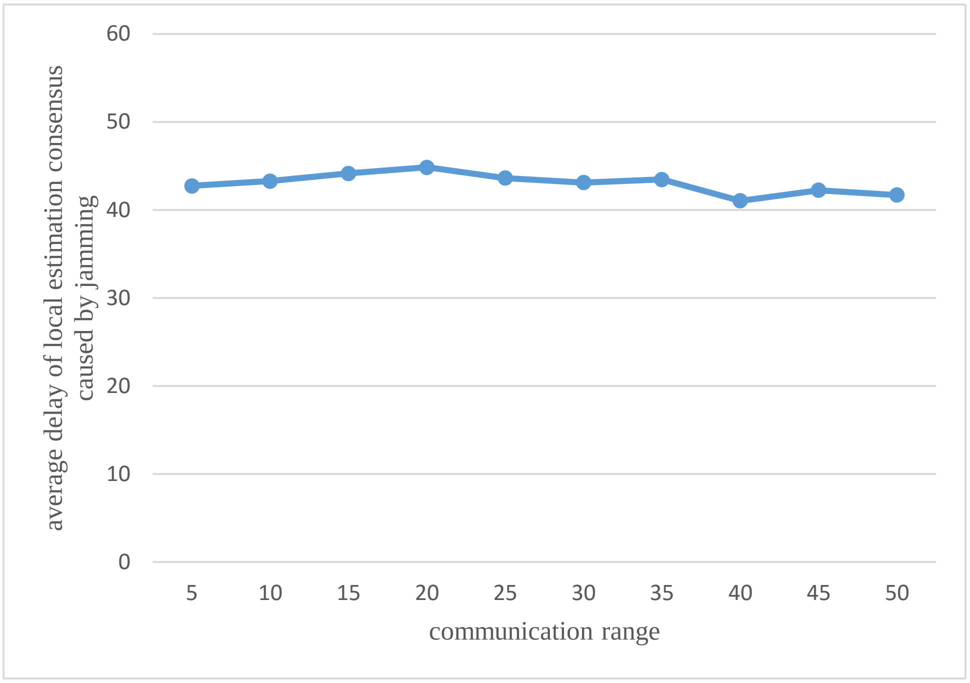

Figure 4.5. Average delay of local estimation consensus caused by jamming with respect to communication range

In summary, our simulation results indicate that, communication jamming delays the achievement of both velocity and estimation consensus. The amount of delay roughly increases linearly with the jamming range. On the other hand, the communication range has no significant impact on the delay of velocity and estimation consensus.

Chapter 5: Communication-aware Formation Control

In this chapter, we consider the formation control problem with the objective of optimizing inter-agent communication performance through achieving the optimum inter-agent distance.

5.1 Methodology

A communication-aware formation control was recently proposed for multi-agent systems with switching topology [20]. It was rigorously proved that the proposed algorithm can optimize the inter-agent communication performance in the multi-agent systems.

In this technique, a communication performance indicator was adopted for formation systems in a practical communication environment, achieving a tradeoff between the antenna far-field and near-field communication. Correspondingly, a new communication-aware formation control law is proposed to maintain the formation and optimize the communication performance.

Consider a multi-agent system consisting of n agents. The dynamics of each agent is given by:

$$\dot{q}_i = u_i \tag{17}$$

where $\dot{q}_i, u_i \in R^2$. $\dot{q}_i$ and u_i denote the position and control input of i_{th} agent.

We denote r_{ij} to be the distance between agent i and agent j:

$$r_{ij} = \left\| q_i - q_j \right\| \tag{18}$$

An approximation reception probability of a SISO communication link is:

$$P(\alpha, \delta, v, r_0, r) = \exp(-\alpha\left(2^{\delta} - 1\right)\left(\frac{r}{r_0}\right)^{v}) \tag{19}$$

where α is a system parameter about antenna characteristic, δ denotes the required application data rate, v is the path loss exponent, r_0 is a reference distance for antenna near-field, and r is the distance between transmitter and receiver.

The reception probability evaluates the probability that the transmitter can influence the receiver. We model the communication channel quality as:

$$a_{ij} = exp\left(-\alpha\left(2^{\delta} - 1\right)\left(\frac{r}{r_0}\right)^{v}\right) \tag{20}$$

Define a set of neighbors of agent i as:

$$N_i = \left\{ j \in v \middle| a_{ij} \leq R \right\} \tag{21}$$

where R is defined as communication range as $R = arg_r\{P(a, \delta, v, r_0, r) = P_T\}$. P_T is a reception probability threshold.

The quality of a SISO reception probability of the receiver decreases when the propagation distance increases [24]. On the other hand, if the transmitter and the receiver are close to each other, the communication would suffer from a lot of interference.

A simple model of antenna near-field communication:

$$g_{ij} = \frac{r_{ij}}{\sqrt{{r_{ij}}^2 + {r_0}^2}} \tag{22}$$

Therefore, we need to find a tradeoff distance, and we define the communication performance indicator as:

$$\phi\left(r_{ij}\right) = \frac{r_{ij}}{\sqrt{{r_{ij}}^2 + {r_0}^2}} \cdot exp\left(-\alpha\left(2^{\delta} - 1\right)\left(\frac{r}{r_0}\right)^{v}\right) \tag{23}$$

In order to optimize the communication performance, a communication-aware formation controller was designed, which evaluates the interaction between neighboring agents. The artificial potential function $\psi(r_{ij})$ is:

$$\psi(r_{ij}) = \phi^* - \phi(r_{ij}), \forall (i,j) \in \varepsilon. \tag{24}$$

Then the communication-aware formation controller is designed as:

$$u_i = -\nabla_{q_i}\left[\sum_{j \in N_i} \psi(r_{ij})\right] = \nabla_{q_i}\left[\sum_{j \in N_i} \phi(r_{ij})\right] \tag{25}$$

The formation controller indicates that agents can move in direction of maximizing the communication performance of neighboring agents. And the gradient of $\phi(r_{ij})$ is computed as:

$$\nabla_{q_i} \phi(r_{ij}) = \rho(r_{ij}) \cdot e_{ij} \tag{26}$$

where $e_{ij} = (q_i - q_j)/r_{ij}$. Thus, we can re-write the communication-aware formation controller as:

$$u_i = \sum_{j \in N_i} \rho(r_{ij}) \cdot e_{ij} \tag{27}$$

5.2 Simulation Results:

In the simulation part, we simulate a group of nine agents to verify the proposed formation control method. The parameters of communication are set as: $\alpha = 10^{-5}$, $\delta = 2$, $n = 3$, $r_0 = 5$, $P_T = 94\%$. The initial positions of nine agents are given by $x_1 = [-5, 16]^T$, $x_2 = [-5, -21]^T$, $x_3 = [1, 1]^T$, $x_4 = [36, -5]^T$, $x_5 = [65, -1]^T$, $x_6 = [70, 10]^T$, $x_7 = [72, -16]^T$, $x_8 = [-5, 0]^T$, $x_9 = [72, 0]^T$. Figures 5.1-5.5 show the initial topology of nine agents and their positions at four subsequent time instants in a 2-D plane. It can be seen from these figures that the system is able to stabilize at a best inter-agent distance among all agents.

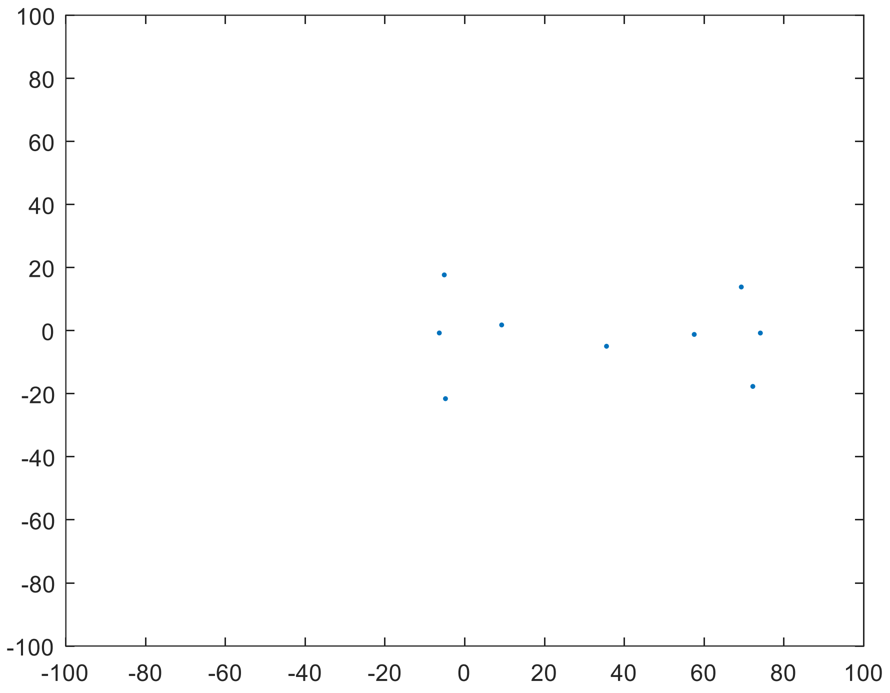

Figure 5.1. The topology of agents at t=0

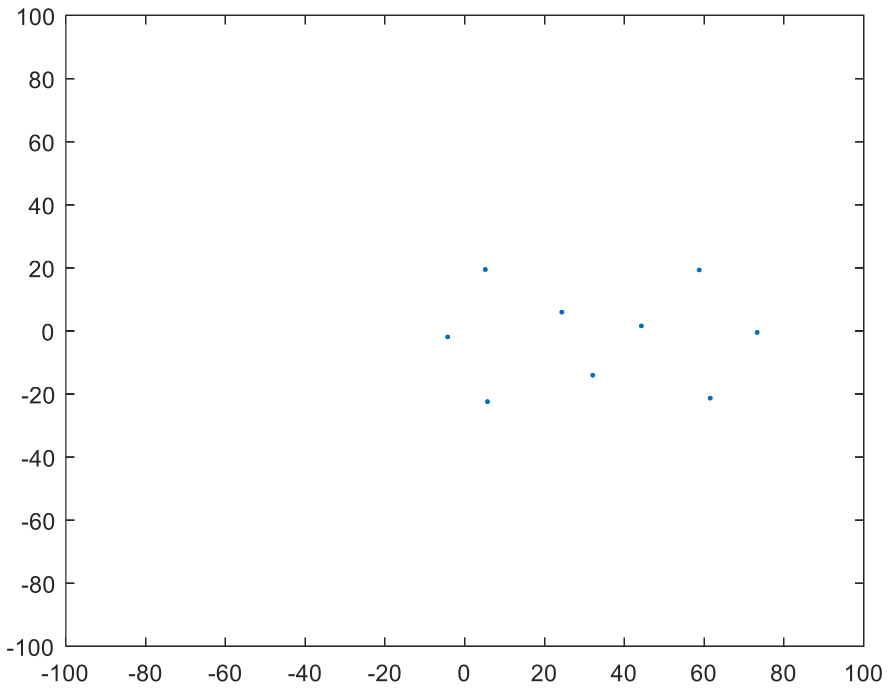

Figure 5.2. The topology of agents at t=3s

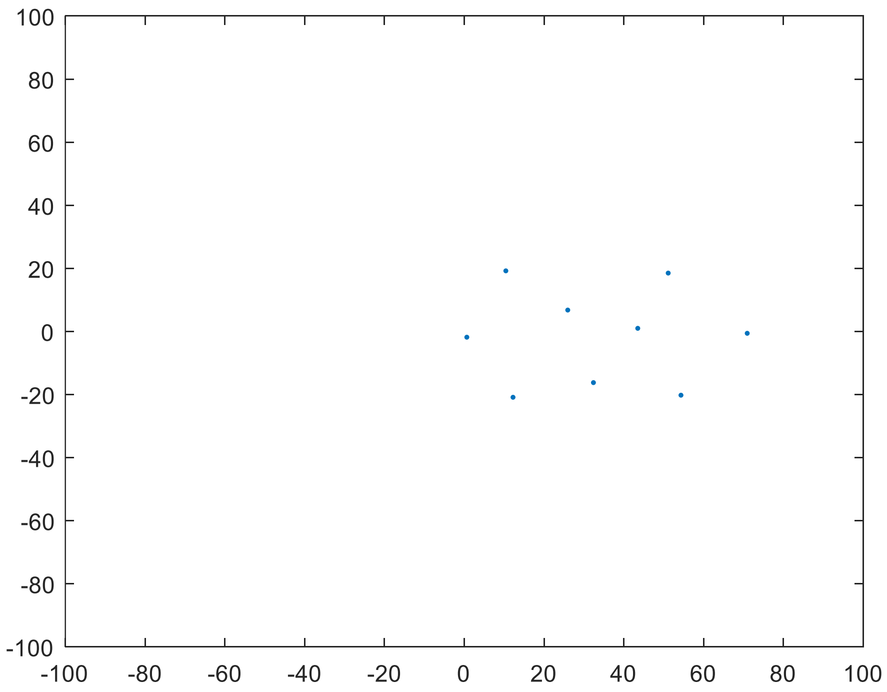

Figure 5.3. The topology of agents at t=6s

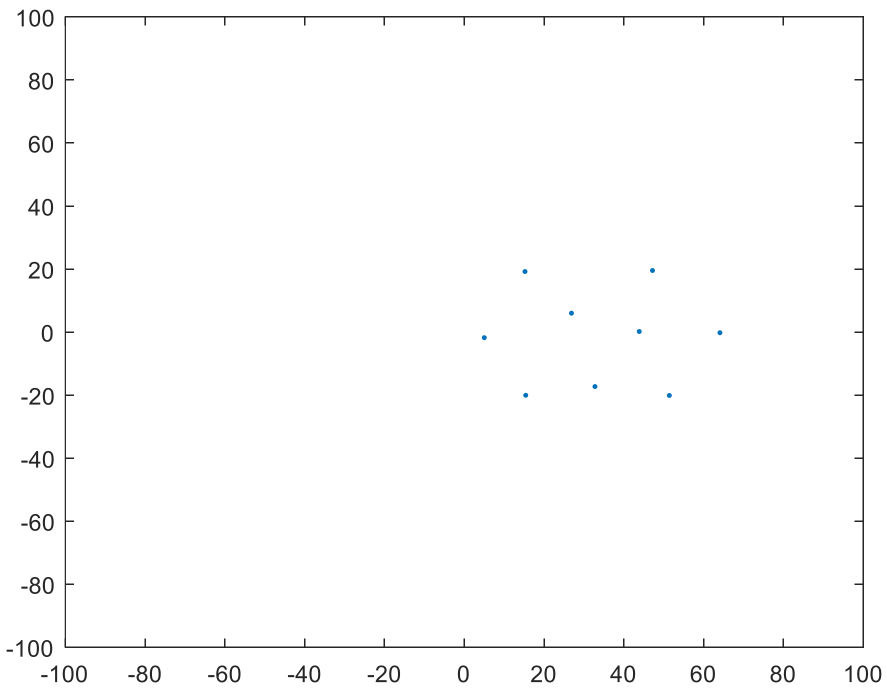

Figure 5.4. The topology of agents at t=9s

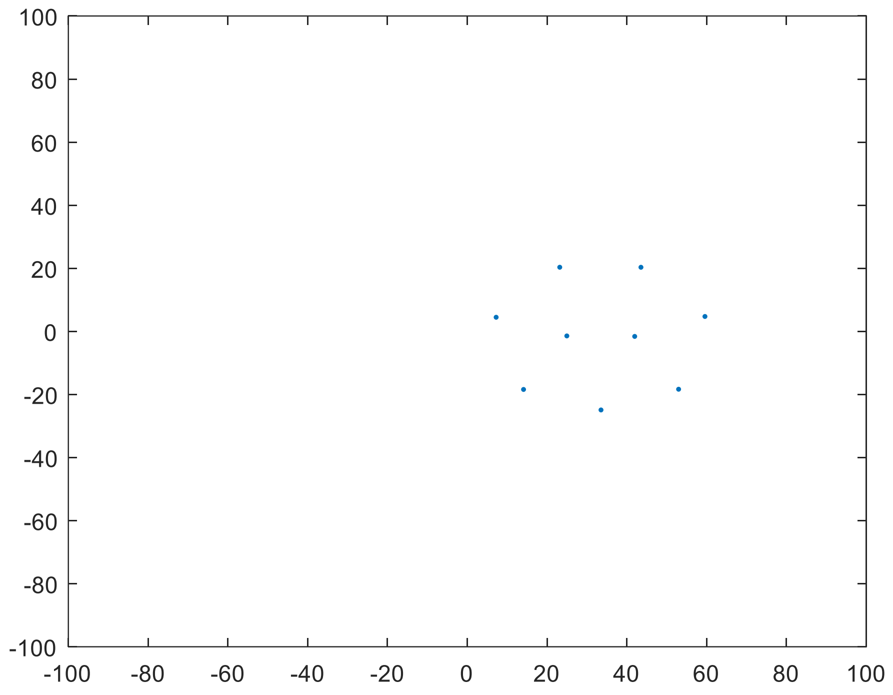

Figure 5.5. The topology of agents at t=12s

Chapter 6: Conclusions and Future Work

6.1 Summary and Conclusions

The main purpose of the research was to investigate communication constraints in distributed multi-agent systems.

First, we implemented a recently proposed distributed global feature estimation algorithm to estimate the position of the multi-agent group's centroid, while adopting a simple flocking model with collision avoidance. The model avoids collision by applying a repelling force moderated by an alignment measure. The algorithm estimates the group centroid through achieving consensus of individual agents' local estimations. To reduce the communication load for information sharing among agents, the implementation of the estimation algorithm adopts an event-triggering mechanism for inter-agent communication. We demonstrated the effectiveness of the proposed estimation algorithm through computer simulations.

Second, we investigated the effect of jamming for the above computer simulations, and measured the delay in achieving consensus for both velocity and distributed estimation. Finally, we studied and verified the effectiveness of a communication-aware formation control strategy through computer simulations, which optimizes communication performance by guiding the system to achieve the optimum inter-agent distance.

6.2 Future Work

In our current study of the distributed estimation algorithm, the event-triggering time sequence is determined by the group error measure, which is not amenable for implementation in a distributed fashion. Our future work will focus on distributed determination of the event-triggering time sequence for each individual agent. Also, the distributed estimation algorithm will be implemented with other popular flocking models, so guidelines for flocking model selection can be developed.

Also, further investigation is to be conducted to study the effects of more sophisticated jamming scenarios, including their impact on communication-aware formation control strategy.

References

[1] C. M. Breder, “Equations Descriptive of Fish Schools and Other Animal Aggregations,” Ecology, vol. 35, pp. 361–370, 1954.

[2] V. Borkar and P. Varaiya, “Asymptotic Agreement in Distributed Estimation,” IEEE Trans. Automatic Control, vol. AC-27, no. 3, pp. 650–655, Jun. 1982.

[3] J. A. Benediktsson and P. H. Swain, “Consensus Theoretic Classification Methods,” IEEE Trans. Sys., Man, Cybern., vol. 22, no. 4, pp. 688–704, Apr. 1992.

[4] D. P. Bertsekas and J. Tsitsiklis, “Parallell and Distributed Computation,” Upper Saddle River, NJ: Prentice-Hall, 1989.

[5] D. P. O’Brien, “Journal of Experimental Marine Biology and Ecology,” 128, (1989) 1.

[6] I. D. Couzin, J. Krause, R. James, G. D. Ruxton and N.R. Franks, “Collective Memory and Spatial Sorting in Animal Groups,” Journal of Theoretical Biology, 218 (1) (2002), pp. 1-11.

[7] I.D. Couzin and N.E. Leonard, “Real-Time Feedback-Controlled Robotic Fish for Behavioral Experiments with Fish Schools,” Proceedings of the IEEE, vol.100, no.1, pp.150-163, Jan. 2012.

[8] F. Cucker and J. Dong, “Avoiding Collisions in Flocks,” IEEE Trans. Automatic Control, vol.55, no.5, pp.1238-1243, May 2010.

[9] F. Cucker and S. Smale, “Emergent Behavior in Flocks,” IEEE Trans. Automatic Control, vol.52, no.5, pp.852-862, May 2007.

[10] M. H. DeGroot, “ Reaching a Consensus,” J. Am. Statist. Assoc., vol. 69, no. 345, pp. 118–121, 1974.

[11] Z. Fu, T. Yang, J. Wang, “Effects of Jamming on a Multi-agent Flocking Model with Distributed Estimation of Global Features”, accepted by FTC 2016, San Francisco, CA, December 2016.

[12] G. Flierl, D. Grünbaum, S. Levin, and D. Olson, “From Individuals to Aggregations: The Interplay Between Behavior and Physics,” J. Theor. Biol., vol. 196, pp. 397–454, 1999.

[13] J. A. Fax and R. M. Murray, “Information Flow and Cooperative Control of Vehicle Formation,” IEEE Trans. Automatic Control, vol. 49, no. 9, pp. 1465–1476, Sep. 2004.

[14] M. A. Goodrich, K. Sean, P. Brian and P. B. Sujit, “What Types of Interactions do Bio-Inspired Robot Swarms and Flocks Afford a Human?” Robotics: Science and Systems (2012).

[15] V. Gazi and K.M. Passino, “Stability Analysis of Swarms,” IEEE Trans. Automatic Control, vol.48, no.4, pp.692-697, April 2003.

[16] D. Grunbaum and A. Okubo, “Modeling Social Animal Aggregations,” Berlin, Germany: Springer-Verlag, 1994, vol. 100, Lecture Notes in Biomathematics, pp. 296–325.

[17] D. Helbing, I. Farkas, and T. Vicsek, “Simulating Dynamical Features of Escape Panic,” Nature, vol. 407, pp. 487–490, 2000.

[18] A. Jadbabaie, J. Lin, and A. S. Morse, “Coordination of Groups of Mobile Autonomous Agents using Nearest Neighbor Rules,” IEEE Trans. Automatic Control, vol. 48, no. 6, pp. 988–1001, Jun. 2003.

[19] N. A. Lynch, “Distributed Algorithms,” San Francisco, CA: Morgan Kaufmann, 1996.

[20] H. Li, J. Peng, W. Liu, J. Wang, J. Liu and Z. Huang, “Flocking Control for Multi-agent Systems with Communication Optimization,” 2013 American Control Conference, Washington, DC, 2013, pp. 2056-2061.

[21] M. Li, I. Koutsopoulos, R. Poovendran, “Optimal Jamming Attack Strategies and Network Defense Policies in Wireless Sensor Networks,” IEEE Trans. Mob. Comput. 9, 1119–1133, 2010.

[22] A. Mogilner and L. Edelstein-Keshet, “A Nonlocal Model for a Swarm,” J. Math. Biol., vol. 38, pp. 534–570, 1999.

[23] P. Ogren, E. Fiorelli, and N. E. Leonard, “Cooperative Control of Mobile Sensor Networks: Adaptive Gradient Climbing in a Distributed Environment,” IEEE Trans. Automatic Control, vol. 49, no. 8, pp. 1292–1302, Aug. 2004.

[24] C. W. Reynolds, “Flocks, Herds and Schools: A Distributed Behavioral Model,” Comput. Graph, vol. 21, Jul. 1987, pp. 25–34.

[25] N. Shimoyama, K. Sugawara, T. Mizuguchi, Y. Hayakawa, and M. Sano, “Collective Motion in a System of Motile Elements,” Phys. Rev. Lett., vol. 76, no. 20, pp. 3870–3873, 1996.

[26] R. O. Saber, J. A. Fax, and R. M. Murray, “Consensus and Cooperation in Networked Multi-agent Systems,” Proceedings of the IEEE, vol. 95, pp. 215–233, 2007.

[27] J. N. Tsitsiklis, “Problems in Decentralized Decision Making and Computation,” Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Lab. Inf. Decision Syst., Massachusetts Inst. Technol., Cambridge, MA, Nov. 1984.

[28] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, “Distributed Asynchronous Deterministic and Stochastic Gradient Optimization Algorithms,” IEEE Trans. Automatic Control, vol. 31, no. 9, pp. 803–812, Sep. 1986.

[29] J. Toner and Y. Tu, “Flocks Herds and Schools: A Quantitative Theory of Flocking,” Phys. Rev. E, vol. 58, no. 4, pp. 4828–4858, Oct. 1998.

[30] S. Vadlamani, B. Eksioglu, H. Medal, A. Nandi, “Jamming Attacks on Wireless Networks: A Taxonomic Survey,” International Journal of Production Economics Volume 172, February 2016, Pages 76–94.

[31] T. Vicsek, A. Cziroók, E. Ben-Jacob, I. Cohen, and O. Shochet, “Novel Type of Phase Transition in a System of Self-driven Particles,” Phys. Rev. Lett., vol. 75, no. 6, pp. 1226–1229, 1995.

[32] J. Wang, I. Ahn, Y. Lu, T. Yang, “A Distributed Detection Algorithm for Collective Behaviors in Multiagent Systems with Unicycle Models”, submitted to International Journal of Control, Automation and Systems, January 2016.

[33] S. C. Weller and N. C. Mann, “Assessing Rater Performance without a Gold Standard using Consensus Theory,” Med. Decision Making, vol. 17, no. 1, pp. 71–79, 1997.

[34] K. Warburton and J. Lazarus, “Tendency-distance Models of Social Cohesion in Animal Groups,” J. Theoret. Biol., vol. 150, pp. 473–488, 1991.

[35] T. Yang, Z. Fu, J. Wang, “Application of an Event-Triggered Distributed Estimation Algorithm in a Simple Multi-agent Flocking Model”, 2016 IEEE SouthEastCon, Norfolk, VA, April 2016.

Appendix: MATLAB Code

```
% Swarm Avoid Collision 2D
% Zhaoyang Fu
% Initialize all parameters
clear all;
clc;
close all;
max_times = 100;
h = 1;
swarm_size = 100;
R=50;
v=1;
theta=2*pi;
H=5;
b=1;
lambda=0;
d=20;
theta_d=2;

% Initialize the position
for i = 1:swarm_size
        swarm(i,1,1)=rand(1)*20;
        swarm(i,1,2)=rand(1)*20;

end

% Initialize the velocity
for j = 1:swarm_size
        v_random1=rand(1);
        swarm(j,2,1)  = v*cos(v_random1*theta);
        swarm(j,2,2)  = v*sin(v_random1*theta);
end

%---------------------------------------------------------

for times = 1 : max_times
% Alignment Measure
for i = 1 : swarm_size
        for j = 1 : swarm_size
            if i>j
              lambda=0;
              lambda=(1/swarm_size)*(lambda+(sqrt((swarm(i, 2, 1)-
swarm(j, 2, 1))^2+(swarm(i, 2, 2)-swarm(j, 2, 2))^2)));
              lambda=sqrt(lambda);     % alignment at a common velocity
is equivalent to lambda=0
            end

        end
end
```

```
for i = 1 : swarm_size
        for j = 1 : swarm_size
            weight_x=0;
            weight_y=0;
            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2)<=R
            weight_a=1;
            else
              weight_a=0;
            end

            weight_x1=0;
            weight_y1=0;
            weight_x1=weight_x1+h*weight_a*(swarm(i, 2, 1)-swarm(j, 2,
1));
            weight_y1=weight_y1+h*weight_a*(swarm(i, 2, 2)-swarm(j, 2,
2));

            weight_x2=0;
            weight_y2=0;
            xi_xj=swarm(i, 1, 1)-swarm(j, 1, 1)^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2;
            f_xij=(xi_xj-d)^(-theta_d);
            if i~=j
            weight_x2=weight_x2+h*f_xij*(swarm(i, 2, 1)-swarm(j, 2, 1));
            weight_y2=weight_y2+h*f_xij*(swarm(i, 2, 2)-swarm(j, 2, 2));
            else
                weight_x2=weight_x2+0;
                weight_y2=weight_y2+0;
            end
            weight_x2=lambda*weight_x2;
            weight_y2=lambda*weight_y2;
        end

        swarm(i, 2, 1)=weight_x1+weight_x2+swarm(1, 2, 1);
        swarm(i, 2, 2)=weight_y1+weight_y2+swarm(1, 2, 2);



            swarm(i,1,1) = swarm(i, 1,1) + h*swarm(i, 2, 1);
            swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);


end
%----------------------------------------------------------
% plot
    clf
    fig=figure(1);
    plot(swarm(:, 1, 1), swarm(:, 1, 2), '.');
    axis([-200 200 -200 200]);
pause(0.1)
end


% Swarm Avoid Collision 3D
% Zhaoyang Fu
% Initialize all parameters
clear;
clc;
close all;
max_times = 50;
h = 1.3;
swarm_size = 100;
R=50;
v=1;
theta=pi;
phi=pi/2;
lambda=0;
d=50;
theta_d=5;

% Initialize the position
index1 = 1;
for i = 1:sqrt(swarm_size)
    for j = 1:sqrt(swarm_size)
        swarm(index1,1,1) = rand(1)*20;
        swarm(index1,1,2) = rand(1)*20;
        swarm(index1,1,3) = rand(1)*20;
        index1 = index1 + 1;

    end
end
% Initialize the velocity
index2= 1;
for i = 1:sqrt(swarm_size)
    for j = 1:sqrt(swarm_size)
        v_random1=rand(1);
        v_random2=rand(1)*(-1)^randi(2);
        swarm(index2,2,1) = v*cos(v_random1*theta)*cos(v_random2*phi);
        swarm(index2,2,2) = v*sin(v_random1*theta)*cos(v_random2*phi);
        swarm(index2,2,3) = v*sin(v_random2*phi);
        index2 = index2 + 1;
    end
end
orig_v(:,2,1)=swarm(:,2,1); % original Vx of swarm
orig_v(:,2,2)=swarm(:,2,2); % original Vy of swarm
orig_v(:,2,3)=swarm(:,2,3); % original Vz of swarm
%----------------------------------------------------------

for times = 1 : max_times
for i = 1 : swarm_size
        for j = 1 : swarm_size
            weight_x=0;
            weight_y=0;
            weight_z=0;
            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2)<=R
            weight_a=1;
            else
              weight_a=0;
            end
            if i>j
              lambda=lambda+1/(swarm_size)*(sqrt((swarm(i, 2, 1)-
swarm(j, 2, 1))^2+(swarm(i, 2, 2)-swarm(j, 2, 2))^2+(swarm(i, 2, 3)-
swarm(j, 2, 3))^2));
            else
              lambda=lambda+0;
            end
            lambda=sqrt(lambda);

            weight_x1=0;
            weight_y1=0;
            weight_z1=0;
            weight_x1=weight_x1+h*weight_a*(swarm(i, 2, 1)-swarm(j, 2,
1));
            weight_y1=weight_y1+h*weight_a*(swarm(i, 2, 2)-swarm(j, 2,
2));
            weight_z1=weight_z1+h*weight_a*(swarm(i, 2, 3)-swarm(j, 2,
3));

            weight_x2=0;
            weight_y2=0;
            weight_z2=0;
            xi_xj=(swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2;
            f_xij=(xi_xj-d)^(-theta_d);
            if i~=j
            weight_x2=weight_x2+h*f_xij*(swarm(i, 2, 1)-swarm(j, 2, 1));
            weight_y2=weight_y2+h*f_xij*(swarm(i, 2, 2)-swarm(j, 2, 2));
            weight_z2=weight_z2+h*f_xij*(swarm(i, 2, 3)-swarm(j, 2, 3));
            else
                weight_x2=weight_x2+0;
                weight_y2=weight_y2+0;
                weight_z2=weight_z2+0;
            end
            weight_x2=lambda*weight_x2;
            weight_y2=lambda*weight_y2;
            weight_z2=lambda*weight_z2;
        end

        swarm(i, 2, 1)=weight_x1+weight_x2+swarm(1, 2, 1);
        swarm(i, 2, 2)=weight_y1+weight_y2+swarm(1, 2, 2);
        swarm(i, 2, 3)=weight_z1+weight_z2+swarm(1, 2, 3);
            swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);
            swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
            swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);

end

%-----------------------------------------------------------
% plot
    clf
    fig=figure(1);
    scatter3(swarm(:, 1, 1), swarm(:, 1, 2),swarm(:, 1, 3) ,'.')
    axis([-200 200 -200 200 -200 200]);
pause(0.1)
end
```

```
% Distributed Estimation
% Zhaoyang Fu
% Initialize all parameters

clear;
clc;
close all;

max_times = 1000;
h =0.1;          %1<=h<=4
swarm_size = 25;  %swarm_size>=2
R=30;             %d<=R<=swarm_size
v=0.1;            %v>0
theta=2*pi;       %theta¡Ê[0, 2pi]
phi=pi;           %phi¡Ê[0, pi]
d=5;             %0<d<R
theta_d=5;        %theta_d>1
R_commu=30;
c=5;
alpha=1;
epsilon_sum=0;

% Initialize the position
for i=1:swarm_size
    swarm(i,1,1) = rand(1)*50;
    swarm(i,1,2) = rand(1)*50;
    swarm(i,1,3) = rand(1)*50;

    Mu_hat_tk(i,1) = 7*swarm(i,1,1); % agent Mu_i(t) x cetriod position
    Mu_hat(i,1)= Mu_hat_tk(i,1);  % agent Mu_i(tk) x cetriod position
    Mu_hat_tk(i,2) = 7*swarm(i,1,2); % agent Mu_i(t) y cetriod position
    Mu_hat(i,2)= Mu_hat_tk(i,2);   % agent Mu_i(tk) y cetriod position
    Mu_hat_tk(i,3) = 7*swarm(i,1,3); % agent Mu_i(t) z cetriod position
    Mu_hat(i,3)= Mu_hat_tk(i,3);  % agent Mu_i(tk) z cetriod position
end

% Initialize the velocity
for j = 1:swarm_size
        v_random1=rand(1);
        v_random2=rand(1);
        swarm(j,2,1)  = v*cos(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,2)  = v*sin(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,3)  = v*sin(v_random2*phi);
end

orig_v(:,2,1)=swarm(:,2,1); % original Vx of swarm
orig_v(:,2,2)=swarm(:,2,2); % original Vy of swarm
orig_v(:,2,3)=swarm(:,2,3); % original Vz of swarm
%---------------------------------------------------------

for times = 1 : max_times

for i = 1 : swarm_size
   t= times*h;
```

```
% Alignment Measure

    for k= 1 : swarm_size
        for j = 1 : swarm_size
            if k>j
              lambda=0;
              lambda=(1/swarm_size)*(lambda+(sqrt((swarm(k, 2, 1)-
swarm(j, 2, 1))^2+(swarm(k, 2, 2)-swarm(j, 2, 2))^2+(swarm(k, 2, 3)-
swarm(j, 2, 3))^2)));
              lambda=sqrt(lambda);   % alignment at a common velocity
is equivalent to lambda=0
            end

        end
    end

% Aviod Collision
            weight_x1=0;
            weight_y1=0;
            weight_z1=0;

            weight_x2=0;
            weight_y2=0;
            weight_z2=0;

            for j = 1 : swarm_size

            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2)<=R
            weight_a=1;
            else
            weight_a=0;
            end

            weight_x1=weight_x1+h*weight_a*(swarm(i, 2, 1)-swarm(j, 2,
1));
            weight_y1=weight_y1+h*weight_a*(swarm(i, 2, 2)-swarm(j, 2,
2));
            weight_z1=weight_z1+h*weight_a*(swarm(i, 2, 3)-swarm(j, 2,
3));



            xi_xj=(swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2;
            f_xij=(xi_xj-d)^(-theta_d);
            if i~=j
            weight_x2=weight_x2+h*f_xij*(swarm(i, 2, 1)-swarm(j, 2, 1));
            weight_y2=weight_y2+h*f_xij*(swarm(i, 2, 2)-swarm(j, 2, 2));
            weight_z2=weight_z2+h*f_xij*(swarm(i, 2, 3)-swarm(j, 2, 3));
            end
            weight_x2=lambda*weight_x2;
            weight_y2=lambda*weight_y2;
            weight_z2=lambda*weight_z2;
            end
        pre_position(i,1)=swarm(i, 1, 1);
        pre_position(i,2)=swarm(i, 1, 2);
        pre_position(i,3)=swarm(i, 1, 3);

        swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);   % update 
position
        swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
        swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);



        swarm(i, 2, 1)=weight_x1+weight_x2+swarm(1, 2, 1); % update 
velocity
        swarm(i, 2, 2)=weight_y1+weight_y2+swarm(1, 2, 2);
        swarm(i, 2, 3)=weight_z1+weight_z2+swarm(1, 2, 3);

        Mu_position(i,1)=swarm(i,1,1)-pre_position(i,1); % 
Mu_i_x=xi(t+h)-xi(t)
        Mu_position(i,2)=swarm(i,1,2)-pre_position(i,2); % 
Mu_i_y=yi(t+h)-yi(t)
        Mu_position(i,3)=swarm(i,1,3)-pre_position(i,3); % 
Mu_i_z=zi(t+h)-zi(t)

            weight_x3=0;
            weight_y3=0;
            weight_z3=0;

            for j = 1 : swarm_size
            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2)<=R_commu
             weight_commu=0.1;
            else
             weight_commu=0;
            end

            if i~=j
            weight_x3=weight_x3+weight_commu*(Mu_hat_tk(j,1)-
Mu_hat_tk(i,1)); % Mu_hat_diff=Mu_hat(j,1)-Mu_hat(i,1);
            weight_y3=weight_y3+weight_commu*(Mu_hat_tk(j,2)-
Mu_hat_tk(i,2));
            weight_z3=weight_z3+weight_commu*(Mu_hat_tk(j,3)-
Mu_hat_tk(i,3));
            end
            end

            Mu_hat(i,1)=weight_x3+Mu_position(i,1)+Mu_hat(i,1); % get 
Mu_hat(t) centroid each time
            Mu_hat(i,2)=weight_y3+Mu_position(i,2)+Mu_hat(i,2);
            Mu_hat(i,3)=weight_z3+Mu_position(i,3)+Mu_hat(i,3);

            epsilon_individual(i)=(Mu_hat(i,1)-
Mu_hat_tk(i,1))^2+(Mu_hat(i,2)-Mu_hat_tk(i,2))^2+(Mu_hat(i,3)-
Mu_hat_tk(i,3))^2;


end
```

```
        epsilon_sum=0;
        for i = 1 : swarm_size
            epsilon_sum=epsilon_individual(i)+epsilon_sum;
        end
            epsilon_swarm=sqrt(epsilon_sum);

            epsilon=c*exp(-alpha*t);
        if epsilon_swarm>epsilon
            for i=1:swarm_size
             Mu_hat_tk(i,1)=Mu_hat(i,1);
             Mu_hat_tk(i,2)=Mu_hat(i,2);
             Mu_hat_tk(i,3)=Mu_hat(i,3);
            end

        end
%---------------------------------------------------------

%     clf
%     fig=figure(1);
%     scatter3(swarm(:, 1, 1), swarm(:, 1, 2),swarm(:, 1, 3) ,'.')
%     axis([-500 500 -500 500 -500 500]);
%     pause(0.1)

    clf
    fig=figure(1);
    scatter3(Mu_hat(:,1), Mu_hat(:,2),Mu_hat(:,3) ,'.')
    axis([-500 500 -500 500 -500 500]);
    pause(0.1)

end
```

```
% Jamming Effect
% Zhaoyang Fu
%----------------------------------------------------------
% Initialize all parameters
clear all;
clc;
close all;

max_times = 1000;
h =0.5;          %1<=h<=4
swarm_size =25;  %swarm_size>=2
R=30;             %d<=R<=swarm_size
v=1;             %v>0
theta=2*pi;       %theta¡Ê[0, 2pi]
phi=pi;           %phi¡Ê[0, pi]
d=1;             %0<d<R
theta_d=2;        %theta_d>1
jam_range=60;
% R_commu=30;
% c=5;
% alpha=1;
% epsilon_sum=0;
%----------------------------------------------------------
% Initialize the position

averagetimes=0;
for k=1:100

for i=1:swarm_size
    swarm(i,1,1) = rand(1)*50-25;
    swarm(i,1,2) = rand(1)*50-25;
    swarm(i,1,3) = rand(1)*50-25;
end
%----------------------------------------------------------
% Initialize the velocity
for j = 1:swarm_size
        v_random1=rand(1);
        v_random2=rand(1);
        swarm(j,2,1)  = v*cos(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,2)  = v*sin(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,3)  = v*sin(v_random2*phi);
end

%----------------------------------------------------------
% set the center of jamming center
jamming_centerx=0;
jamming_centery=0;
jamming_centerz=0;
% for i=1:swarm_size
%     jamming_centerx=swarm(i,1,1)+jamming_centerx;
%     jamming_centery=swarm(i,1,2)+jamming_centery;
%     jamming_centerz=swarm(i,1,3)+jamming_centerz;
% end
% jamming_centerx=(1/swarm_size)*jamming_centerx;
% jamming_centery=(1/swarm_size)*jamming_centery;
% jamming_centerz=(1/swarm_size)*jamming_centerz;
```

```
%----------------------------------------------------------
for times = 1 : max_times

%    t= times*h;
% Alignment Measure
lambda=0;
for i = 1 : swarm_size
        for j = 1 : swarm_size
            if i>j
            lambda=lambda+((swarm(i, 2, 1)-swarm(j, 2, 1))^2+(swarm(i,
2, 2)-swarm(j, 2, 2))^2+(swarm(i, 2, 3)-swarm(j, 2, 3))^2);
            end
            lambda=sqrt((1/swarm_size)*lambda);    % alignment at a
common velocity is equivalent to lambda=0
        end
end

for i = 1 : swarm_size

%----------------------------------------------------------
% jamming and Aviod Collision



% Aviod Collision

         distance=sqrt((swarm(i, 1, 1)-jamming_centerx)^2+(swarm(i, 1,
2)-jamming_centery)^2+(swarm(i, 1, 3)-jamming_centery)^2);

         if distance>jam_range
            weight_x1=0;
            weight_y1=0;
            weight_z1=0;

            weight_x2=0;
            weight_y2=0;
            weight_z2=0;

            for j = 1 : swarm_size

            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2)<=R
            weight_a=1;
            else
            weight_a=0;
            end

            weight_x1=weight_x1+h*weight_a*(swarm(i, 2, 1)-swarm(j, 2,
1));
            weight_y1=weight_y1+h*weight_a*(swarm(i, 2, 2)-swarm(j, 2,
2));
            weight_z1=weight_z1+h*weight_a*(swarm(i, 2, 3)-swarm(j, 2,
3));
```

```
            xi_xj=(swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2;
            f_xij=(xi_xj-d)^(-theta_d);
            if i~=j
            weight_x2=weight_x2+h*f_xij*(swarm(i, 1, 1)-swarm(j, 1, 1));
            weight_y2=weight_y2+h*f_xij*(swarm(i, 1, 2)-swarm(j, 1, 2));
            weight_z2=weight_z2+h*f_xij*(swarm(i, 1, 3)-swarm(j, 1, 3));
            end

            weight_x2=lambda*weight_x2;
            weight_y2=lambda*weight_y2;
            weight_z2=lambda*weight_z2;
            end

        swarm(i, 2, 1)=weight_x1+weight_x2+swarm(1, 2, 1); % update
velocity
        swarm(i, 2, 2)=weight_y1+weight_y2+swarm(1, 2, 2);
        swarm(i, 2, 3)=weight_z1+weight_z2+swarm(1, 2, 3);

        swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);   % update
position
        swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
        swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);

         else
          swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);   % update
position
          swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
          swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);
         end

end

    figure(1);
    scatter3(swarm(:, 1, 1), swarm(:, 1, 2),swarm(:, 1, 3), '.');
%    axis([-1000 1000 -1000 1000 -1000 1000]);
    figure(2);
    plot(times,lambda,'.');
    hold on;
    pause(0.1);


    if lambda<=0.041
     lambda
        averagetimes=averagetimes+times;
    break
    end

end

end
averagetimes=1/k*averagetimes;
```

```
% Jamming effect on centroid estimation
% Zhaoyang Fu
%---------------------------------------------------------
% Initialize all parameters
clear all;
clc;
close all;

max_times = 1000;
h =0.5;         %1<=h<=4
swarm_size =25;  %swarm_size>=2
R=30;            %d<=R<=swarm_size
v=1;           %v>0
theta=2*pi;      %theta¡Ê[0, 2pi]
phi=pi;          %phi¡Ê[0, pi]
d=1;            %0<d<R
theta_d=2;       %theta_d>1
jam_range=60;
% R_commu=30;
% c=5;
% alpha=1;
% epsilon_sum=0;
%---------------------------------------------------------
% Initialize the position

averagetimes=0;
for k=1:100

for i=1:swarm_size
    swarm(i,1,1) = rand(1)*50-25;
    swarm(i,1,2) = rand(1)*50-25;
    swarm(i,1,3) = rand(1)*50-25;
end
%---------------------------------------------------------
% Initialize the velocity
for j = 1:swarm_size
        v_random1=rand(1);
        v_random2=rand(1);
        swarm(j,2,1) = v*cos(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,2) = v*sin(v_random1*theta)*cos(v_random2*phi);
        swarm(j,2,3) = v*sin(v_random2*phi);
end

%---------------------------------------------------------
for times = 1 : max_times

%    t= times*h;
% Alignment Measure
lambda=0;
for i = 1 : swarm_size
        for j = 1 : swarm_size
            if i>j
            lambda=lambda+((swarm(i, 2, 1)-swarm(j, 2, 1))^2+(swarm(i,
2, 2)-swarm(j, 2, 2))^2+(swarm(i, 2, 3)-swarm(j, 2, 3))^2);
            end
            lambda=sqrt((1/swarm_size)*lambda);    % alignment at a
common velocity is equivalent to lambda=0
        end
end



for i = 1 : swarm_size

%----------------------------------------------------------
% jamming and Aviod Collision



% Aviod Collision

         distance=sqrt((swarm(i, 1, 1)-jamming_centerx)^2+(swarm(i, 1,
2)-jamming_centery)^2+(swarm(i, 1, 3)-jamming_centery)^2);

         if distance>jam_range
            weight_x1=0;
            weight_y1=0;
            weight_z1=0;

            weight_x2=0;
            weight_y2=0;
            weight_z2=0;

            for j = 1 : swarm_size

            if sqrt((swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2)<=R
            weight_a=1;
            else
            weight_a=0;
            end

            weight_x1=weight_x1+h*weight_a*(swarm(i, 2, 1)-swarm(j, 2,
1));
            weight_y1=weight_y1+h*weight_a*(swarm(i, 2, 2)-swarm(j, 2,
2));
            weight_z1=weight_z1+h*weight_a*(swarm(i, 2, 3)-swarm(j, 2,
3));



            xi_xj=(swarm(i, 1, 1)-swarm(j, 1, 1))^2+(swarm(i, 1, 2)-
swarm(j, 1, 2))^2+(swarm(i, 1, 3)-swarm(j, 1, 3))^2;
            f_xij=(xi_xj-d)^(-theta_d);
            if i~=j
            weight_x2=weight_x2+h*f_xij*(swarm(i, 1, 1)-swarm(j, 1, 1));
            weight_y2=weight_y2+h*f_xij*(swarm(i, 1, 2)-swarm(j, 1, 2));
            weight_z2=weight_z2+h*f_xij*(swarm(i, 1, 3)-swarm(j, 1, 3));
            end
            weight_x2=lambda*weight_x2;
            weight_y2=lambda*weight_y2;
            weight_z2=lambda*weight_z2;
            end

        swarm(i, 2, 1)=weight_x1+weight_x2+swarm(1, 2, 1); % update
velocity
        swarm(i, 2, 2)=weight_y1+weight_y2+swarm(1, 2, 2);
        swarm(i, 2, 3)=weight_z1+weight_z2+swarm(1, 2, 3);

        swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);   % update
position
        swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
        swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);

         else
          swarm(i,1,1) = swarm(i, 1, 1) + h*swarm(i, 2, 1);   % update
position
          swarm(i,1,2) = swarm(i, 1, 2) + h*swarm(i, 2, 2);
          swarm(i,1,3) = swarm(i, 1, 3) + h*swarm(i, 2, 3);
         end

end

    figure(1);
    scatter3(swarm(:, 1, 1), swarm(:, 1, 2),swarm(:, 1, 3), '.');
    figure(2);
    plot(times,lambda,'.');
    hold on;
    pause(0.1);


    if lambda<=0.041
     lambda
        averagetimes=averagetimes+times;
    break
    end

end

end
averagetimes=1/k*averagetimes;
```

```
% Communication-aware formation control
% Zhaoyang Fu
clear all;
clc;
close all;

% Initialize all parameters
max_times = 1000;
h =1;
swarm_size=7;
theta=2*pi;
alpha=10^(-5); % system parameter about antenna characteristics
delta=2; % required application data rate
Beta=alpha*(2^delta-1);
v=3; % path loss exponent
r0=5; % reference antenna near-field
PT=0.94;  % reception probability threshold
u=1;
rho_ij=0;

%swarm=[-5,14;-5,-19;0,0;35,-4;68,0;72,13;72,-18]

swarm=zeros(swarm_size,2);
        swarm(1,1)=-5;
        swarm(1,2)=16;
        swarm(2,1)=-5;
        swarm(2,2)=-21;
        swarm(3,1)=1;
        swarm(3,2)=1;
        swarm(4,1)=36;
        swarm(4,2)=-5;
        swarm(5,1)=65;
        swarm(5,2)=0;
        swarm(6,1)=70;
        swarm(6,2)=10;
        swarm(7,1)=72;
        swarm(7,2)=-16;
        swarm(8,1)=-5;
        swarm(8,2)=0;
        swarm(9,1)=72;
        swarm(9,2)=0;

% Initialize the velocity
for j = 1:swarm_size
        u_random1=rand(1);
         speed(j,1) = 0;
         speed(j,2) = 0;
        %speed(j,1) = u*cos(u_random1*theta);
        %speed(j,2) = u*sin(u_random1*theta);
        %speed(j,1) = u*cos(theta);
        %speed(j,2) = u*sin(theta);
end

for k=1:max_times

for i=1:swarm_size
```

```
    rho_ij=0;
    for j=[1:(i-1),(i+1):swarm_size]
        rij=sqrt((swarm(i,1)-swarm(j,1))^2+(swarm(i,2)-swarm(j,2))^2);
        aij=exp(-alpha*(2^delta-1)*(rij/r0)^v);

        if aij>=PT
            rho_ij=(-Beta*v*rij^(v+2)-
Beta*v*(r0^2)*(rij^v)+r0^(v+2))*exp(-
Beta*(rij/r0)^v)/sqrt((rij^2+r0^2)^3);
        else
            rho_ij=0;
        end
        qi=[swarm(i,1),swarm(i,2)];
        qj=[swarm(j,1),swarm(j,2)];
        nd=(qi-qj)/norm(qi-qj)

        speed(i,1)=speed(i,1)+rho_ij*nd(1);
        speed(i,2)=speed(i,2)+rho_ij*nd(2);
    end
            swarm(i,1)=swarm(i,1)+speed(i,1)*h
            swarm(i,2)=swarm(i,2)+speed(i,2)*h
            speed(i,1)=0;
            speed(i,2)=0;
end


    clf
    fig=figure(1);
    plot(swarm(:, 1), swarm(:, 2), '.');
    axis([-100 100 -100 100]);
    pause(0.1)
    end
```