2007-06

Linear and planar array formation in wireless sensor networks

Gkionis, Charalampos

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

LINEAR AND PLANAR ARRAY FORMATION IN WIRELESS SENSOR NETWORKS

by

Charalampos Gkionis

June 2007

Thesis Advisor:	Murali Tummala
Thesis Co-Advisor:	John McEachen
Second Reader:	T. Owens Walker III

Approved for public released; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Linear and Planar Antenna Array in Wireless Sensor Networks		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Charalampos Gkionis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

Wireless sensor networking (WSN) is a relatively new field of research with many applications, both military and commercial. In the military applications, WSNs could be used in hostile environments to minimize the need for human presence. A WSN consists of a large number of small sensor nodes that are deployed in an area of interest for collecting information. A subgroup of nodes then collaborate their transmissions to achieve beamforming. The information collected by the WSN is relayed to an unmanned aerial vehicle (UAV), which is synchronized with the transmission beam of the network. In this study, the positioning of the nodes in a WSN is investigated with the main object to propose a method to find the best combination of nodes for beamforming given a random distribution in the sensor field. Additionally, the method is expandable in two dimensions and capable of forming a planar antenna array which will improve the beamforming gain. A simulation model was developed in MATLAB code to study the formation of linear and planar antenna array of nodes. The existing iterative technique in the formation of a linear antenna array is compared with the proposed and the results showed an improvement in linearity.

14. SUBJECT TERMS sensor networks, array formation, distributed sensor network			**15. NUMBER OF PAGES** 124
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public released; distribution is unlimited

LINEAR AND PLANAR ARRAY FORMATION IN WIRELESS SENSOR NETWORKS

Charalampos Gkionis
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author: Charalampos Gkionis

Approved by: Murali Tummala
Thesis Advisor

John McEachen
Thesis Co-Advisor

T. Owens Walker III
Second Reader

Jeffrey Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Wireless sensor networking (WSN) is a relatively new field of research with many applications, both military and commercial. In the military applications, WSNs could be used in hostile environments to minimize the need for human presence. A WSN consists of a large number of small sensor nodes that are deployed in an area of interest for collecting information. A subgroup of nodes then collaborate their transmissions to achieve beamforming. The information collected by the WSN is relayed to an unmanned aerial vehicle (UAV), which is synchronized with the transmission beam of the network. In this study, the positioning of the nodes in a WSN is investigated with the main object to propose a method to find the best combination of nodes for beamforming given a random distribution in the sensor field. Additionally, the method is expandable in two dimensions and capable of forming a planar antenna array which will improve the beamforming gain. A simulation model was developed in MATLAB code to study the formation of linear and planar antenna array of nodes. The existing iterative technique in the formation of a linear antenna array is compared with the proposed and the results showed an improvement in linearity.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my utmost gratitude to Professor Murali Tummala, CDR T. Owens Walker III and Professor John McEachen of Naval Postgraduate School, Monterey, California for their guidance and contribution to the successful completion of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

A wireless sensor network (WSN) consists of large number of small sensor nodes that are deployed over an area in order to collect information. The military applications that can be used include monitoring of biological, nuclear, chemical weapons, terrorist attacks actions and reconnaissance. The information collected by the sensor nodes must be transferred to an analysis center for further investigation and decision-making. The sensor nodes, due to their limited power capabilities, have limited transmission range. By providing distributed power usage across the network, the nodes can coordinate their transmissions through beamforming and increase the transmission range. The beam of the network is coordinated with an unmanned aerial vehicle (UAV), and the data collected by the sensor nodes are transmitted to the UAV.

The objective of this thesis was to select appropriate nodes in a randomly deployed sensor field to form a linear or planar array for beamforming. For the WSN to perform distributed beamforming and information transmission, a suitable subgroup of nodes must be selected from the cluster. The subgroup of nodes selected should satisfy specific criteria in inter-node spacing and linearity. Multiple methods for both linear and planar array formation were proposed.

For the linear array formation, we presented three methods: iterative, concurrent and line fitting approach; and for the planar array formation, two methods: concurrent and line fitting. The iterative approach begins with three nodes and then expands to the desired number of nodes of the array. The concurrent approach is a technique based on the iterative approach and the line fitting approach is based on the solutions found by constructing and rotating a line obtained based on least squares line fitting in a cluster. Expanding the concurrent and the line fitting approaches, we proposed two methods for planar array formation.

To evaluate the performance of each method, we introduce a set of metrics used for calculating the errors of the array formed. For the linear array, we proposed three

error metrics: perpendicular distance error, inter-node spacing error and total error; and for the planar array, we used one error metric: total error.

A simulation model was developed and implemented in MATLAB with multiple node densities in the sensor field, communications range of each node and communication frequencies as parameters. The array performance was evaluated using the defined error metrics.

Results showed that the proposed methods demonstrate an improvement in linearity in the construction of a linear array when compared with existing techniques. The density of nodes in the sensor field plays a significant role in reducing the error metrics in both linear and planar arrays. The average error metrics in the construction of the linear arrays decreased as the density of nodes increased for all simulations. Additionally, for the same operating frequency, and as more nodes were added to the array, the total error decreased slightly with density for the line fitting method.

The linear array formation is successfully extended to the planar array, which demonstrated improved performance. The line fitting method is used in all cases and the total error decreased when the node density in the area was increased.

I. INTRODUCTION

Over the years, wireless sensor nodes have developed into low cost, small size devices with multiple capabilities. The vision is that thousands of these smart microsensors can be deployed on a battlefield by an aircraft. After a self-configuration process among the nodes, the wireless sensor network (WSN) will be capable of collecting and analyzing the information throughout the network. The collected data will be transmitted to end users located outside the network for further analysis and decision-making.

Due to the limited power capabilities of each node, the transmission range will be very small. Also, the nodes will be distributed in a non-friendly environment in terms of the surface and obstacles, and therefore it is likely that the radio horizon of each node will be limited. A new method is proposed by [1] and [2] to distribute power usage across the sensor network, which enables a better and broader spread of energy consumption among the nodes (see Figure 1).

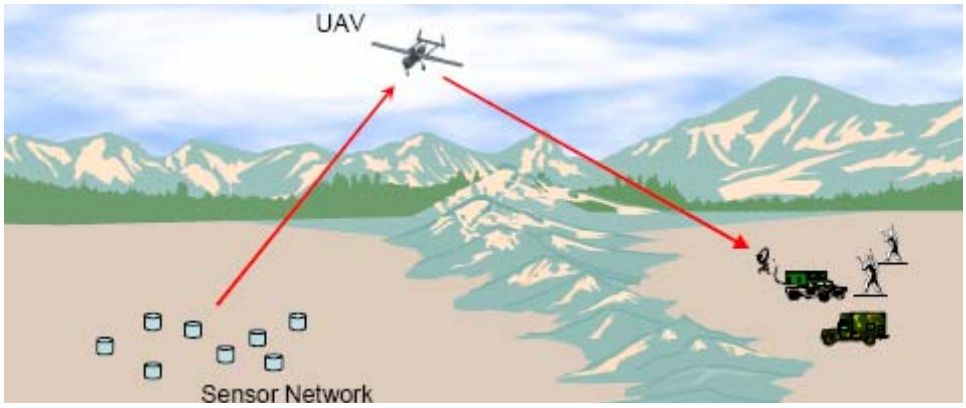

Figure 1. Use of a UAV in distributing power usage of the sensor network (From Ref.[1])

In distributing the power usage of the sensor network, nodes organize into a subgroup (known as a transmit cluster), coordinate their transmissions and form a beam, which has the effect of increasing transmission range. The beam is then aligned with the receiver of an unmanned aerial vehicle (UAV), which flies above the network according

to a pre-determined search plan, and the information of the network is relayed to the command and control point as depicted in Figure 2.

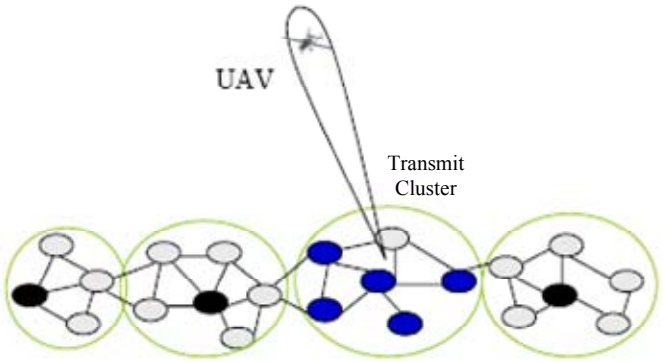

Figure 2. Transmit Cluster and Beamforming in WSN with Nodes Coordinating Transmission (From Ref. [1])

A. THESIS OBJECTIVE

The objective of this thesis is to propose a method to find the best combination of nodes for beamforming, given a random distribution in the sensor field. The method should select a specific set of nodes from the field that closely approximates the best possible array for beamforming. The nodes selected should be aligned as close as possible to a linear array with inter-node distances of one-half a wavelength. The method should be expandable to two dimensions for forming a planar array, which will improve the beamforming gain.

A two part process in forming the most appropriate arrays of nodes is proposed in this work. The first part consists of a concurrent and a line fitting approach based on least squares regression method to construct a linear array. The purpose is to improve the beamforming gain achieved compared to the iterative construction of an array as presented in [1]. The second part involves expanding the array constructed using the line fitting approach to a planar, two-dimensional array. Both linear and planar simulation models have been developed in MATLAB and their results are compared.

B. RELATED WORK

A beamforming approach for distributed wireless sensor networks introduced by Vincent et al. [1] assembles a subset of sensor nodes into a distributed array for

beamforming. A distributed array is constructed by specific nodes, and the data gathered by the network are transmitted as a narrow beam toward an overhead UAV. The performance of the technique used to construct the distributed array, referred to as iterative approach, in [1] is compared to the line fitting approach proposed and developed in this thesis (in chapter III).

Distributing power usage in wireless sensor networks for energy conservation is presented by Vincent et al. in [1] and [3]. A method to reassign the transmit cluster that forms (along with other nodes of the network) the distributed array is presented, achieving three main goals. First, a broader spread of the energy consumption, second, minimizing the energy expended in moving the cluster and, finally, reducing to the extent practicable the time to bring the UAV and the sensor network's beam into alignment.

Lee et al. [4] presents algorithms for forming specific geometric patterns with a mobile wireless sensor network without the assistance of the user. The patterns presented are a line, circle and regular polygon. The nodes forming the specific patterns do not exchange information but are capable of mobility and are equipped with cameras on board. These two characteristics can improve the ability to construct specific patterns of nodes in the network but also require increased processing and moving capabilities. These capabilities lead to an increased power consumption, which is not desired in a sensor field covering a 24/7 military surveillance area.

Arrayed wireless sensor networks and the problem of forming an array is presented by Elissaios et al. in [5]. Wireless arrays are the dynamic entities that are formed by specific groups of nodes under specific rules. The main advantage for wireless array, as described, is their capability to suppress interference, localize nodes and also act as message forwarding agents. The method proposed focuses on the selection of nodes for the formation of wireless arrays.

Lastly, collaborative beamforming for distributed wireless ad hoc sensor networks is analyzed by Ochiai et al. in [6] using the theory of random arrays. It is shown that with N sensor nodes uniformly distributed in an area, the directivity can approach N, provided that the nodes are located sparsely enough.

C. THESIS OUTLINE

The organization of the thesis is as follows. Chapter II introduces the wireless sensor networks with respect to various architectures, standards in use, routing techniques, positioning, and localization of nodes as well as an introduction to array beamforming. Chapter III presents the proposed methods for the construction of linear and planar arrays. Chapter IV presents the performance evaluation metrics and simulation results of the constructed grids, for both the linear and planar arrays. Finally, in Chapter V, the overall conclusions and the highlights of future work are discussed. The appendix includes the MATLAB source code used in the simulation studies.

II. WIRELESS SENSOR NETWORKS AND BEAMFORMING

This chapter presents an overview of WSN, including definitions, characteristics, limitations, architecture and clustering in WSN. Next, the method for distributing power usage across a WSN, proposed by [1], is reviewed. The method suggests collaboration of the nodes in the network and the use of a UAV as an intermediate for transferring the information to an analysis center. Finally, distributed beamforming as performed by wireless sensor nodes is discussed.

A. OVERVIEW OF WSN

WSNs are expected to have a tremendous impact in the near future both for military and commercial applications. Sensor networks represent a significant improvement over traditional sensors, networks and wireless communications. In this section, we will present the basic definitions, characteristics and limitations in WSNs, and we will address the most important design factors.

1. Definitions and Description

A WSN is a wireless network consisting of a large number of small-sized sensor nodes with short range communication range, deployed either into the phenomenon of interest or very close to it. The nodes are autonomous devices, which cooperate with each other in transferring information and are spatially distributed in a sensor field.

The nodes consist of a small microprocessor, a battery, a radio transceiver and one or more sensors in accordance with the specific sensing tasks that the network has to accomplish. Sensors monitor physical, environmental or human actions, such as temperature, pressure, motion, various gasses, and vibration. The sensor variety is only limited by the node size and the capability of the network to process, transfer and analyze the information within a reasonable time delay.

Applications of WSNs include military, environmental, acoustic, seismic, medical monitoring, and also fire, motion, object detection. The WSN is distributed in the area where it is assigned to monitor or detect a specific phenomenon.

2. Characteristics and Limitations

The main design characteristics of WSNs that affect their overall performance are energy efficiency, data routing efficiency, fault tolerance, delay and throughput, scalability, synchronization and localization.

Limitations of WSN include the size of the nodes, which should be kept as small as possible, especially for military applications when monitoring an area. Size limits the storage and memory capabilities of the nodes. The cost of sensor nodes is also a limitation since the number of nodes in the field may range from hundreds to thousands. Both size and cost of the nodes contribute to the major constraint in a WSN, which is limited energy resources due to the limited battery life. Communication range is affected and limited due to this as well.

The nodes have to communicate and participate in the sensing task in an energy efficient manner with low power consumption for the network to be deployed and work effectively. The information needs to be routed with the following constraints in mind: power consumption for a single packet, reliability and quality of the network, protocol efficiency depending on the application, number of hops among source and nodes, time required for the communication, and need for adaptation to the environment and to specific network conditions.

In WSNs, routing is one of the most significant functions and needs to be designed efficiently. Routing protocols that meet the needs of most of the criteria above can be classified as either proactive or reactive routing protocols [7]. In the design of a WSN, the selection of a suitable routing protocol is based on the specific purpose that the network has to accomplish.

a. Energy Efficiency

Energy efficiency is another major concern in the performance of WSNs. Since each node can only be equipped with a limited power source (typically less than 0.5 Ah, 1.2 V) [8], the power must be well managed among communication and computation processes. The node's lifetime depends on the battery lifetime, excluding manufacture or physical malfunctions.

Thus, power conservation and management should be thoroughly addressed when designing a WSN. The power consumption can be divided into three categories: sensing, data processing, and data communication [8]. Among the three, data communication has the highest percentage of power consumption. Also, while the first two are determined when designing the hardware of the node, the communication power consumption is highly dependent on the specific conditions of the environment (i.e., positioning, location, etc.) and the transmission medium (i.e., weather conditions) which cannot be accurately predetermined.

b. Density

The number of the sensor nodes in the network field depends on the specific application for which the WSN is deployed. The density can vary from a few nodes to hundreds or thousands in a region. The density μ can be calculated as

$$\mu(R) = \frac{N \pi R^2}{A}, \qquad (1)$$

where N is the number of nodes in a region of area A and R is the radio transmission range [9]. So, $\mu(R)$ gives the number of nodes within the transmission radius of each node in region A.

3. Architecture and Clustering

Flat and tiered are the two main architectures that exist for wireless sensor networks. In flat sensor networks, the nodes are homogenous in functionality and capabilities while tiered sensor networks have the basic view of a pyramid as in Figure 3. In a tiered network, some nodes may have more capabilities and functionality than others. The higher nodes in the hierarchy provide services to those below. For large sensor networks to be able to function correctly, it has been suggested that clustering is required [10]. The main categories of tiered sensor networks are: geographical, information, and security clustering [6].

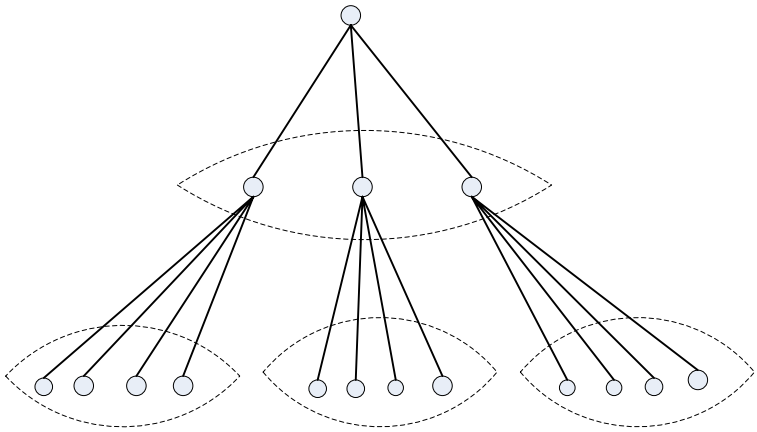

Figure 3. Tiered Architecture in Low Rate Wireless Personal Area Networks (LR-WPAN) (from [11])

This thesis is focused on a flat network of homogeneous nodes that organize themselves into *clusters* based on certain conditions, depending on the application. Each cluster will select a single node as a coordinator or *clusterhead* (CH).

The CH, in our study, will have increased responsibilities but not increased physical capabilities and will be charged with managing the resources within the cluster as well as maintaining communication with neighboring clusters. The selected CH may be any of the nodes within the network and the selection is based only on the sensor field local density criteria (i.e., different areas (spots) in the sensor field would have different density due to random distribution of nodes). The basic mode of organizing the sensor network in this task is the geographical clustering in which the criterion for selecting the cluster is based on geographical proximity of the neighbor nodes.

The communication distance for each node is proportional to the output power; in other words, the minimum output power required to transmit a signal over a distance d is proportional to d^n, where $2 \le n < 4$ (n is closer to 4 for a low-lying antenna and near-ground channels) [8], [12]. Since the nodes in a wireless sensor network are sensitive to energy consumption, the communication range for each node will be low. In [8], it is shown that the received power starts to drop with higher exponents at smaller distances for low antenna heights.

For a sensor node, the communication range will be a circle with radius r. In this area, the Signal to Noise and Interference ratio (SNIR) of the signal received by any node

should be above the receiver threshold. The receiver threshold depends only on the manufacturing quality and characteristics of the node's receiver. Keeping in mind that the sensor nodes are made simply and inexpensively and also the limitations in the output power of the transmitter, the communication range of a node should be considered low for simulating realistic scenarios.

4. Deployment

The topology establishment of a WSN can be divided into three main phases [8]:

- Pre-deployment and deployment phase
- Post-deployment phase
- Redeployment of additional nodes phase

In the pre-deployment and deployment phase, the sensor nodes are deployed in the field. This can be achieved by either positioning them one by one or by dropping them from the air by a UAV, a helicopter or an aircraft.

In the second phase, right after deployment of the sensor nodes in the field, the network has to organize itself according to the specific task to be performed. Any topology changes in this phase are due to malfunctions, node position, communication range capabilities and task details.

The last of the three deployment phases mentioned above is required in order to maintain the initial node density in the field to replace the malfunctioning or failed nodes.

B. METHOD FOR DISTRIBUTING POWER USAGE ACROSS A WSN [1]

WSN are especially attractive for military applications because they provide unattended surveillance of the deployed area. A WSN can be deployed into a hostile environment and gather information, replacing a vulnerable group of soldiers. No human needs to be exposed to the threat (e.g., into a chemically effected environment). The nodes, instead, can gather all the information needed and transmit them to a distributed analysis center.

Each node, though, has limited transmission power, so the transmission distance is also limited. Other factors that may limit the transmission distance are the surface terrain or the presence of obstacles. By combining the transmissions of a subgroup of nodes through beamforming, the transmission distance can be significantly increased. This method of distributing the power usage of the WSN addresses the transmission of the data over larger distances through efficient management of the nodes' available power in the absence of an absolute positioning system.

Without GPS, the nodes cannot determine their absolute position and must rely on inter-node distances. In [1], to distribute power usage across a WSN it is assumed that these distances between nodes are known. The network, given the inter-node distances in the cluster, is capable in forming an array as will be described in Chapter III. The additional task is that the network will have to determine all of inter-node distances among the nodes that are in communication distance of each other. The inter-node distance can be calculated by measuring propagation delay of radio signals among the nodes.

1. Tasks Performed in Post Deployment Phase of a Distributing Power Usage WSN

The second phase or post-deployment phase is when the network must perform its functions which, in our case, are the following:

- An initial CH must be assigned according to an algorithm.
- Each node has to determine its distance among other neighbor nodes.
- Nodes are organized into clusters, and selected nodes must form a specific array pattern (one or two-dimensional array).
- Data are transmitted outside the network by coordinating the transmission and forming a beam.

In order for the sensor network field to be able to find its CH, a beacon message is sent from every node after the initial deployment. Each node transmits a beacon message, and the message is received by its neighbors. Additionally, each node receives its

neighbor's messages. For example node i with N_i neighbors receives N_i beacon messages. The node with the most neighbors in its communication distance is chosen to be the CH.

After determining the CH in a deployment area, the information gathered from the sensors in the network needs to be transferred to a distributed analysis center. This analysis center is assumed to be away from the hostile environment where the network has been deployed. It may be a secure military base hundreds of miles away for safety purposes. The basic objective is the fast and reliable transfer of the information from the sensor network into a secure distributed center [1] for analysis purposes.

To accomplish this, a UAV is flown above the wireless sensor nodes for a limited time period, long enough for the network to transmit the information collected. Using a subgroup of nodes that belong to the cluster, a *transmit cluster* [1], which is able to transmit the information to the UAV, is formed (see Figure 2). The information gathered in the WSN is transmitted to the UAV by the participation of the specified nodes in the transmit cluster, which coordinate their transmissions in order to form a beam. The beam of the network is engaged and synchronized with the UAV, and the transmission of the data initializes. The entire process does not include any human interaction until the data is received by the distributed analysis center. The UAV serving as an aerial relay between the WSN and the analysis center is shown in Figure 1.

Because of the limited on duty time of the UAV in the hostile environment, the transmission from the sensor network needs to be fast and reliable. The signals transmitted by the subset of nodes to the UAV are identical to the signals used for inter-node communication except for pre-calculated phases and amplitude offsets needed to perform the beamforming. The theoretical gain achieved after beamforming is proportional to the square of the number of participating nodes ($\propto N^2$ where N is the number of participating nodes) [1],[13]. For a fixed Signal to Noise Ratio (SNR) for the communication link between the network and the UAV, the more nodes participating in the array formation, the less power is required and the better is the beamforming gain achieved. However, even for a large N, the transmission of data between the sensor

network and the UAV is energy consuming. Thus, after a certain time, a change in the specified CH and subgroup of nodes may be required.

After entering the sensor network area, the UAV has to search for the sensor network beam. Several methods could be used for fast engagement and synchronization of the UAV. For the alignment of the beam towards the UAV, two search plans exist, the progressive search and the random search. These techniques are widely used in antisubmarine warfare for a uniform search in an area for a stationary target [14]. For a random search, it is proven that, on average, we cover the entire area before detecting the target, but also the probability of detection is never assured. In the progressive search, we expect to cover only half the area before detecting the target. However, assuming independent segments of the search is not realistic. The probability of detection in any given minute depends on whether or not we detected the target during the previous minute [3] [14].

C. BEAMFORMING

In the post deployment phase of a WSN, a method as presented above, for distributing the power usage of the sensor network could be achieved by combining the transmission of selected nodes of a subgroup. Since each node has an omni-directional antenna and limited transmission capabilities, only by combining the transmission of multiple nodes together can we achieve greater transmission distances. The selected nodes would coordinate their transmissions by transmitting the same signal with calculated phase and amplitude offsets [1]. The electromagnetic waves will interfere and the total radiated power would be focused in a predetermined direction (e.g., UAV). The concentrated power in the preferred direction is the gain G and the participating nodes for achieving this gain are the array elements. As more nodes participate in the array, we are able to achieve a higher gain. Two array geometries are typically considered: linear (one dimensional) and planar (two dimensional) [3], [15].

1. Linear Array Beamforming

In this section, we will study the beamforming achieved by M omni-directional elements. The M elements are assumed to be in a line with equal spacing among them. We assume that a source exists in the far field at an angle θ_α with respect to x (array) axis. The source is transmitting a signal $s(t)$ modulating a complex carrier $e^{j\omega_c t}$ and we assume that the arriving wavefront is planar [15] as shown in Figure 4.

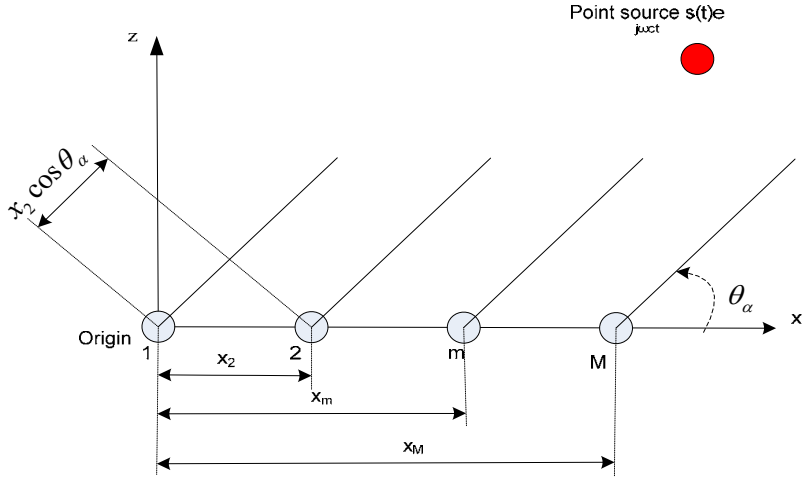

Figure 4. An $M \times 1$ Uniform Linear Antenna Array

The m^{th} element receives the signal

$$t_m(\theta_a) = \frac{x_m \cos\theta_\alpha}{c} \tag{2}$$

seconds before it is received by the first element at the origin, where c is the speed of light and x_m is the distance from the origin to the m^{th} element. The sum of the antenna outputs is the array factor (spatial response $F(\theta)$) of the array

$$F(\theta_a) = \sum_{m=1}^{M} e^{j\omega_c t_m(\theta_a)} \quad \text{or} \quad F(\theta_a) = \sum_{m=1}^{M} e^{j\frac{2\pi}{\lambda} x_m \cos(\theta_a)} \tag{3}$$

where λ is the wavelength of the source.

If we desire to point the radiated power in a specific direction, we have to multiply the output of each array element by a complex weight, $I_m e^{-j\omega_c t_m(\theta_0)}$, where θ_0 is the angle of arrival measured from the array axis. The main lobe of the radiation pattern

is designed to give the maximum value of $F(\theta_a)$ at $\theta_a = \theta_0$. The squared magnitude of the radiation power pattern is given by

$$G(\theta_a) = k\left|F(\theta_a)\right|^2, \tag{4}$$

where k is a proportionality constant [15].

2. Planar Array Beamforming

Figure 5 shows an $M \times N$ planar array located in the far field of a point source [15]. The array factor in the two-dimensional case is given by

$$F(\theta,\phi) = \sum_{m=1}^{M} e^{j\frac{2\pi}{\lambda}(x_n \sin\theta\cos\phi + y_n \sin\theta\cos\phi)}, \tag{5}$$

where (x_n, y_n) are the coordinates of the n^{th} point, ϕ is the azimuth angle with respect to $x-$axis, θ is the elevation angle with respect to $z-$axis and the array is assumed to be uniformly excited.

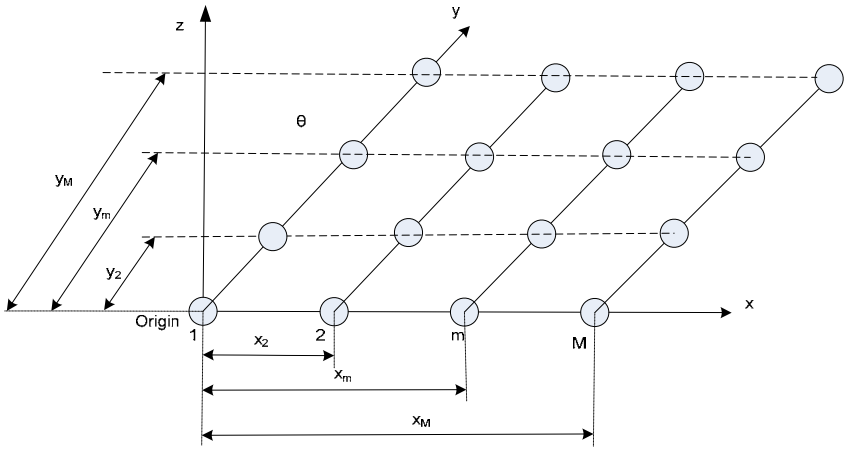

Figure 5. An $M \times N$ Antenna Array of Omni-directional Radiating Elements [15]

D. SUMMARY

In this chapter, we presented the main definitions and characteristics of a WSN and also addressed the constraints in design and deployment. Next, we described a method for distributing power usage across the WSN in a military environment as

presented by [1] using a UAV for transferring the data of the network to an outlying analysis center. Finally, the basics of beamforming in one- and two-dimensions were briefly presented.

The next chapter will present the proposed methods for forming these linear and planar arrays in a WSN.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROPOSED METHODS FOR BEAMFORMING ARRAY FORMATION IN WIRELESS SENSOR NETWORKS

This chapter presents the proposed array formation techniques. We describe the techniques used by the network to organize the nodes of a cluster through inter-node distance estimation to form linear and planar arrays to achieve increased gain.

To perform beamforming, as described in Chapter II, the antenna elements must be in specific positions. On the other hand, the nodes are randomly deployed in the WSN. Therefore, the network should be capable of self determining which nodes would be selected to participate in this group.

For the WSN to perform distributed beamforming and information transmission, a suitable subgroup of nodes must be selected from the cluster. For linear array formation, we present three methods: iterative, concurrent and line fitting and for planar array formation, two methods: concurrent and line fitting. All algorithms include a CH in the subgroup of nodes that perform the beamforming. The parameters that affect the array formation and the shapes that the nodes (as array elements) form will be analyzed.

In a distributed wireless sensor network, the nodes are able to determine their position if a subset of nodes initially has knowledge of their absolute geographical coordinates. The idea is that neighboring nodes measure their distances to the beacon nodes, and transmit their positions to their neighbors [3]. New nodes then become beacon nodes and transmit their position in an iterative fashion. Applications that depend on precise positioning information cannot be relied upon to guarantee performance in critical military scenarios due to the error introduced in positioning information. In the techniques presented, the network does not need to construct an absolute coordination system for each node prior to forming an array. The only knowledge that each node is assumed to have is the distance to all of its neighbors within the communication radius [3].

A. LINEAR ARRAY FORMATION

In linear array formation, three approaches are presented: iterative, concurrent and line fitting.

The iterative approach is a technique proposed by [1], which initially begins with three nodes and then expands to the desired number of nodes of the array. The concurrent approach is a technique based on the iterative approach and the line fitting approach is based on the solutions found by constructing and rotating a line obtained based on least squares line fitting in a cluster.

For our case, in order to perform beamforming (as described in the previous chapter), we desire the inter-node distance to be $\lambda/2$, where λ is the wavelength ($\lambda = c/f$). Two criteria need to be satisfied: first, the nodes need to be in a line and, second, the spacing between adjacent nodes needs to be $\lambda/2$.

1. Linear Array Formation using the Iterative Approach

In this procedure, as suggested in [3], the nodes are examined in groups of three. Consider three nodes S, i and j (S being the CH) and relative distances among them $d(S,i)$, $d(i,j)$ and $d(S,j)$. As shown in Figure 6, the three nodes are not aware of their absolute positions (meaning, they do not know whether they are located in a plane as in Figure 6 (a) or Figure 6 (b)). However, the nodes do know the distances among themselves and to all of those nodes that they can communicate with.

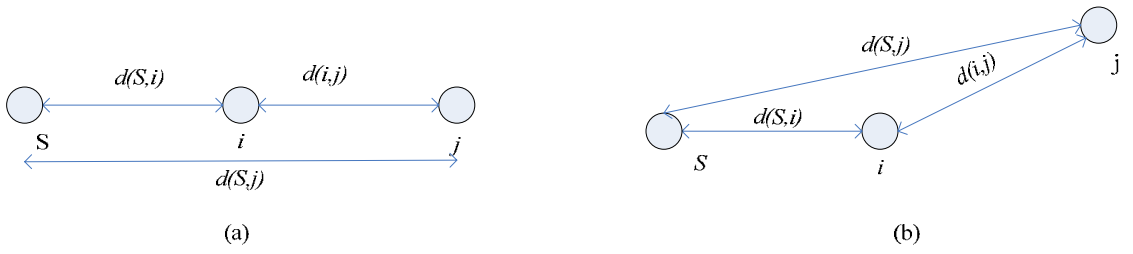

Figure 6. Relative Orientation of the Nodes [3]

To select the most linear shape among the three nodes, an algorithm is used and described in [3] that defines two constraints that must be met for a linear array. First, the distances among the closest nodes need to be equal. For our example in Figure 6,

$$d(S,i) = d(i,j) \tag{6}$$

which leads to the following objective:

$$\varepsilon_1 = \min\{|\, d(S,i) - d(i,j)\,|\}\,. \tag{7}$$

Second, the sum of the distances of the closest nodes (inter-node spacing distance) must be equal to the distance between the first and the last node. Again for our example,

$$d(S,i) + d(i,j) = d(S,j) \tag{8}$$

which leads to the following objective:

$$\varepsilon_2 = \min\{|\, d(S,i) + d(i,j) - d(S,j)\,|\} \tag{9}$$

Combining (7) and (9) and adding appropriately chosen weights, α and β, gives

$$\varepsilon_3 = \min\{\alpha\,\big|d(S,i) + d(i,j) - d(S,j)\big| + \beta\big|d(S,i) - d(i,j)\big|\}\,. \tag{10}$$

By minimizing (10), best three nodes along a line with approximately equal distances among them are chosen.

In order to satisfy the criterion that the spacing between adjacent nodes be $\lambda/2$, we minimize (10) to get

$$\varepsilon_4 = \min\{\alpha\,\big|d(S,i) + d(i,j) - \lambda\big| + \beta(|\, d(S,i) - \frac{\lambda}{2}\,| + |\, d(i,j) - \frac{\lambda}{2}\,|)\} \tag{11}$$

Using the above, the first three nodes are chosen with the best linear characteristics and with an approximate inter-node spacing of $\lambda/2$. Next, a fourth node is iteratively added using the technique described above for three nodes. The last two nodes in one end or the other of the first three nodes are used as reference points to evaluate the new node. Finally, by proceeding iteratively, we expand the array to the total desired number of nodes.

2. Linear Array Formation using the Concurrent Approach

Based on the previously described technique, we now propose a new solution to finding nodes placed in a linear shape. The solution solves the problem in a concurrent manner for the desired number of nodes and also takes into account the desired inter-node spacing.

For solving the problem concurrently, we assume that from a uniformly distributed random sensor field, we want to select the nodes that are positioned in as straight a line as possible. In Figure 7, we present two examples of nodes that are not suitable for beamforming. In Figure 7 (a), the nodes are not in a straight line, and, in Figure 7 (b), the nodes are not equally spaced. We need to find a group of nodes that meet both criteria.

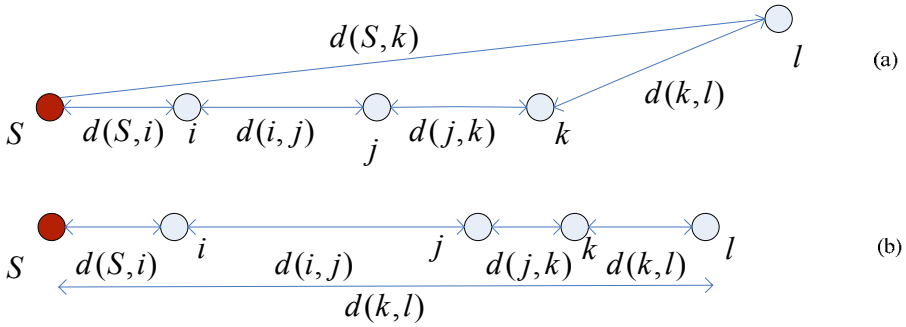

Figure 7. Five Nodes Not Suitable for Beamforming

Marked in red (S) in Figure 7, the CH, as in the previous method, must be included in the group. The remaining four nodes are i, j, k and l. To meet the first criterion (i.e., the nodes are in a line), we minimize the equation

$$\varepsilon_{C_1} = | d(S,i) + d(i,j) + d(j,k) + d(k,l) - d(S,l) | . \qquad (12)$$

By minimizing (12), we can be assured that the nodes selected are those that are positioned as close as possible on a straight line. By this, we are trying to eliminate the situation where the nodes are positioned as in Figure 7 (a).

Our algorithm must also choose the nodes that are equally spaced with distance $\lambda/2$, i.e., we desire

$$d(S,i) = \frac{\lambda}{2},\ d(i,j) = \frac{\lambda}{2},\ d(j,k) = \frac{\lambda}{2},\ d(k,l) = \frac{\lambda}{2} \text{ and } d(S,l) = 4 \times \frac{\lambda}{2} . \qquad (13)$$

This leads to a minimization of

$$|d(S,i)-\frac{\lambda}{2}|=|d(i,j)-\frac{\lambda}{2}|=|d(j,k)-\frac{\lambda}{2}|=|d(k,l)-\frac{\lambda}{2}|=|d(S,l)-4\times\frac{\lambda}{2}| \tag{14}$$

Finally, combining the elements of (14), we need to minimumize

$$\begin{aligned}\varepsilon_{C_2} == \{&|d(S,i)-\frac{\lambda}{2}|+|d(i,j)-\frac{\lambda}{2}|+|d(j,k)-\frac{\lambda}{2}|+\\ &|d(k,l)-\frac{\lambda}{2}|+|d(S,l)-4\times\frac{\lambda}{2}|\}\end{aligned} \tag{15}$$

Combining (12) and (15) and adding appropriate weights, we have

$$\begin{aligned}\varepsilon_C = \alpha\{&|d(S,i)+d(i,j)+d(j,k)+d(k,l)-d(S,l)|\}+\\ \beta\{&|d(S,i)-\frac{\lambda}{2}|+|d(i,j)-\frac{\lambda}{2}|+|d(j,k)-\frac{\lambda}{2}+|d(k,l)-\frac{\lambda}{2}|+|d(S,l)-4\times\frac{\lambda}{2}|\}\end{aligned} \tag{16}$$

At last, we have achieved both criteria (i.e., positioning five nodes as closely as possible in a straight line with inter-node spacing among adjacent nodes at $\lambda/2$). The "best case" theoretical positioning result is shown in Figure 8.

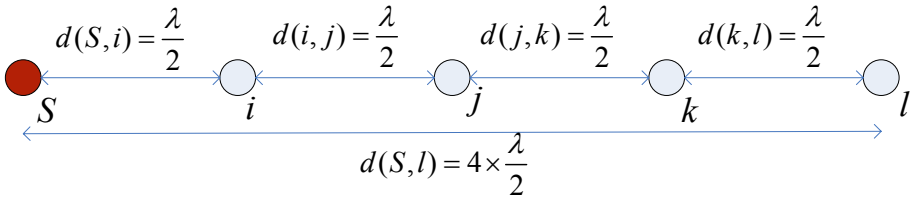

Figure 8. Five Node Ideal Linear Array with Inter-node spacing $\lambda/2$ Suitable for Beamforming

The concurrent method described above solves for the formation of an array in a one step process, which means that we solve the problem for the total desired number of nodes in one step. If *N* nodes are needed to form the array, then the solution is solved initially for *N* nodes; in the iterative method, on the other hand, we initially solve for three nodes and then expand to *N*. This approach would examine all the possible combinations within the cluster and the solution that minimizes (16) would be selected.

3. Linear Array Formation using the Least Squares Line Fitting Approach

The least squares fitting technique (regression line) is a statistical procedure for finding the best linear fit to a set of points by minimizing the sum of the squares of differences between the points generated by the function and corresponding points in the data (residuals). The linear regression function which provides the linear solution is given by

$$f(x) = c_2 x + c_1 \tag{17}$$

where c_2 is the slope of the line and c_1 the offset. The node positions and the linear regression solution are shown in Figure 9.

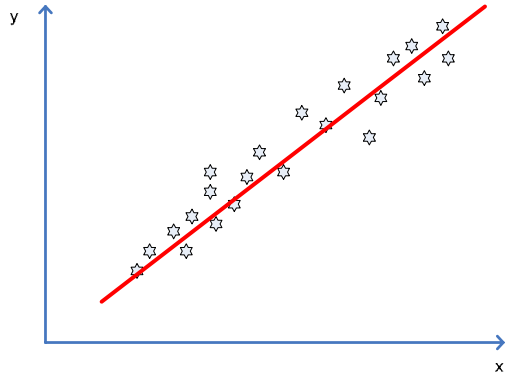

Figure 9. The Least Squares Line Fitting Solution

The least squares line fitting is applied to all the nodes that are within the communication distance of the CH (i.e., cluster). The result is Line γ shown in Figure 10. The CH does not necessarily lie on the line, but it should clearly be very close to it since it is typically positioned near the center of the cluster.

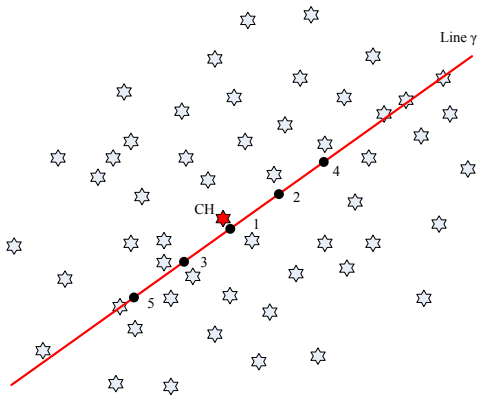

Figure 10. Least Squares Line Fitting applied to Cluster

The heavy dot numbered 1 on Line γ in Figure 10 is the orthogonal projection of the CH onto the line (i.e., intersection of the line from CH perpendicular to Line γ). The heavy dots numbered 2 and 3 are spaced exactly $\lambda/2$ from the first on Line γ and so on. These dots represent the ideal node positions along the Line γ and their inter-node spacing can be summarized as

$$d(s_1, s_2) = d(s_2, s_4) = d(s_1, s_3) = d(s_3, s_5) = \frac{\lambda}{2}. \qquad (18)$$

where $d(s_1, s_2)$ is the distance between Dot 1 and Dot 2 along the line.

In order to have a linear array consisting of nodes equally spaced by $\lambda/2$, the theoretically perfect solution would be the dots on the least squares fitted line. In practice, in a randomly deployed (uniformly distributed) sensor network field, we select the sensor nodes that have the minimum distance from the dots. The solution is shown in the Figure 11, where the nodes marked in green are selected as the closest nodes to Dots 2 to 5. The CH and Nodes 2-5 form the linear array for beamforming.

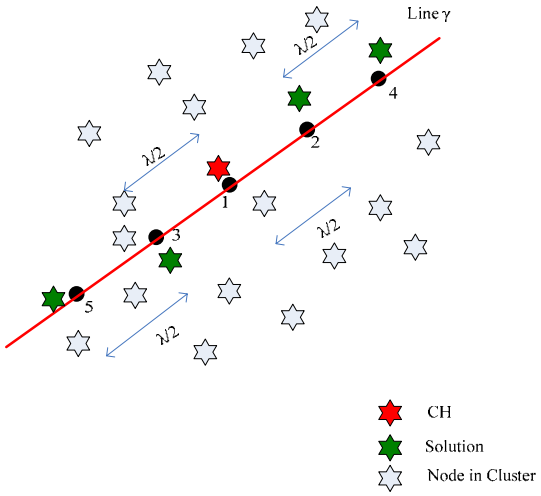

Figure 11. Formation of the Linear Array using Least Squares Error Fitting

In an attempt to form the best possible linear array centered around the CH, (the least squares fitted) Line γ is rotated in counter clockwise direction by an angle φ as shown in Figure 12. After the rotation, again four ideal node locations with inter-node distances of $\lambda/2$ are marked as heavy dots along the rotated line. Followed by that, four sensor nodes closest to the new ideal locations (dots) are selected (marked in blue). The

two solutions (green and blue nodes) are compared (according to metrics described in the next chapter) and the one with the least error is chosen to form the array.

The least squares line fitting and iterative approaches are simulated in Matlab. Figure 13 shows the results of simulation for a 7-node array. The green lines represent two solutions for the least squares approach after a rotation of 90^0 degrees and the red line represents the solution of the iterative method. Both methods are implemented in a field of 10×10 meters for 500 nodes. Only the nodes in the communication radius of the CH are shown.

While the least squares line fitting is applied initially to the cluster area, an alternative method would be to place a line randomly through the CH, and then rotate it multiple times using the above process. The initial direction of the line applied to the cluster does not effect the final solution. The number of the rotations depends on the density of the nodes and the exact number would need to be found heuristically. In a high density field, more rotations would be required in order to examine all the possible solutions.

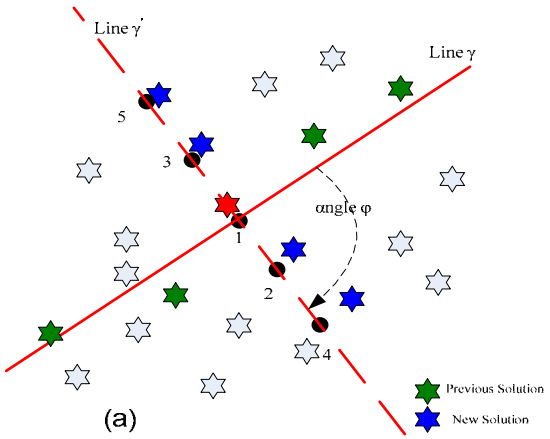

Figure 12. Formation of Linear Array using Least Squares Line Fitting after Rotation by an Angle φ

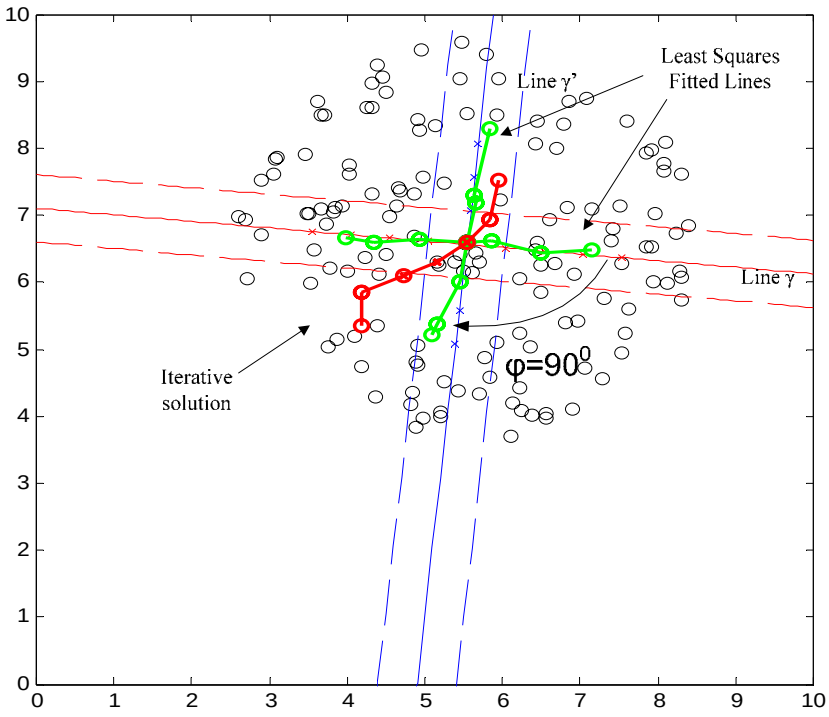

Figure 13. Comparison of the Line Fitting (green line) Solution with the Iterative Construction (red line) in MATLAB in a sensor field of size $10\times10\ m^2$ with 500 nodes

B. PLANAR ARRAY FORMATION

Expanding the techniques described for the formation of a linear array, we can form an array in two dimensions (i.e., planar array). Again, the larger the size of the array with inter-node spacings based on $\lambda/2$, the better is the beamforming gain. Here, planar array on a square grid (i.e., 3×3, 4×4, 5×5, etc.) are considered.

Both the concurrent and the line fitting approach are discussed below for planar array formation.

1. Planar Array Formation using Concurrent Approach

Based on the concurrent technique presented in Section A, we expand it to the construction of a planar array (in two dimensions) for 9 nodes. The concept is to use the

distance-based equations to describe the shape of the structure required. For increased beamforming gain, we require more nodes to participate in the array as elements.

For constructing a planar array with 9 nodes we require them to be positioned as shown in Figure 14 (ideal square shape). The CH may be in one of three possible positions (marked with red filled circle).

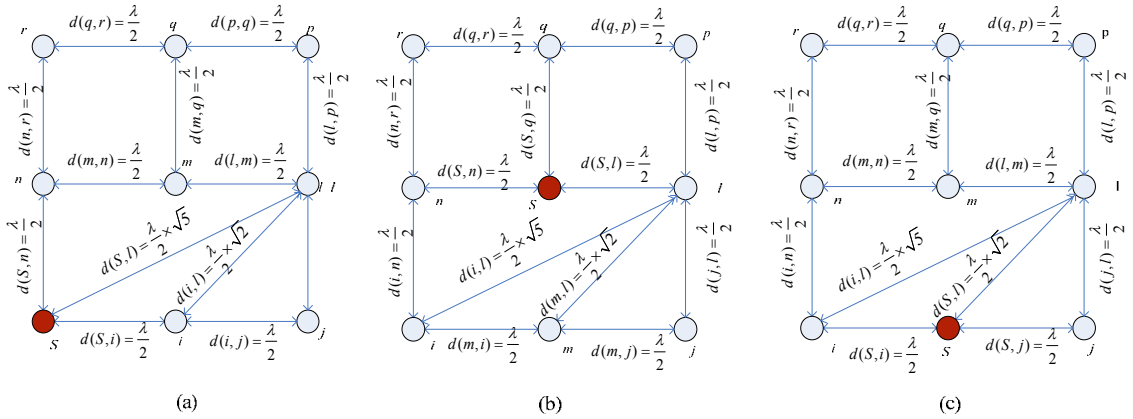

Figure 14. Inter-node distances of Planar Array: Possible Positions of the CH in red and the Participating Nodes are hollow circles

Any other combination in the construction of a 3×3 planar array would be a mirror image of one of the arrays in Figure 14. This is shown in Figure 15, where the solution marked with blue nodes has three mirror images (marked with green) which equate to the same solution set in the field. For each of the three possibilities, we calculate the error metric and choose the one with the least error.

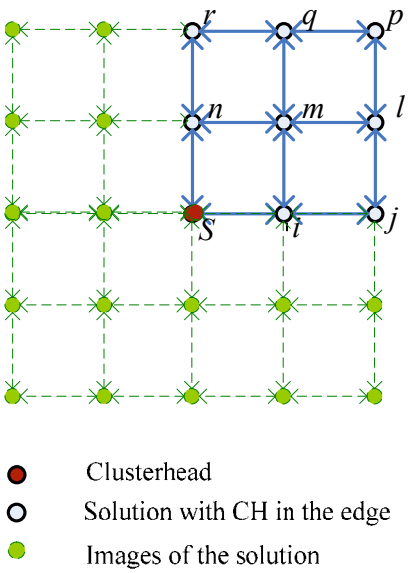

Figure 15. Solution and mirror images of the solution in the field

To form a planar array, we have to find the inter-node distances for each node pair. In the case of the CH in Figure 14 (a), the various distances for a perfect planar array are as shown in the Figure 16.

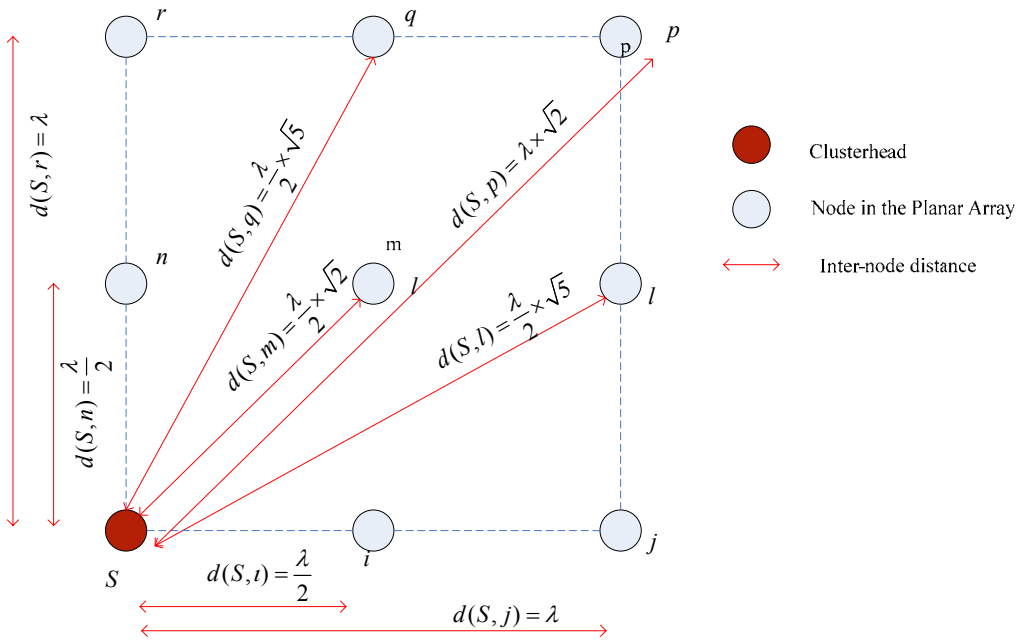

Figure 16. Node Distances from the CH in a 3×3 Array

Solving only for inter-node distances between the CH and all other nodes, we have the following equation to be minimized

$$\begin{aligned}\varepsilon_{CH} = &| d(S,i)-\frac{\lambda}{2}| + | d(S,j)-\lambda | + | d(S,l)-\frac{\lambda}{2}\times\sqrt{5} | + | d(S,m)-\frac{\lambda}{2}\times\sqrt{2} | + \\ &| d(S,n)-\frac{\lambda}{2}| + | d(S,p)-\lambda\times\sqrt{2} | + | d(S,q)-\frac{\lambda}{2}\times\sqrt{5} | + | d(S,r)-\lambda |\end{aligned} \tag{19}$$

Each factor of the above equation must be minimized; the sum of all factors must be optimized by again solving for the minimum.

Following the same procedure for the remaining nodes, we have

$$\varepsilon = \min | \varepsilon_{CH} + \varepsilon_i + \varepsilon_j + \varepsilon_l + \varepsilon_m + \varepsilon_n + \varepsilon_p + \varepsilon_q + \varepsilon_r | . \tag{20}$$

where ε_{CH} is the inter-node error to be minimized for distances between CH and every other node in the array, ε_i is likewise the inter-node error to be minimized with Node i as the anchor node, and so on. Combining all the inter-node error returns, we have the total inter-node error to be minimized as given by

$$\begin{aligned}\varepsilon = &| d(S,i)-\frac{\lambda}{2}| + | d(S,j)-\lambda | + | d(S,l)-\frac{\lambda}{2}\times\sqrt{5} | + | d(S,m)-\frac{\lambda}{2}\times\sqrt{2} | + \\ &| d(S,n)-\frac{\lambda}{2}| + | d(S,p)-\lambda\times\sqrt{2} | + | d(S,q)-\frac{\lambda}{2}\times\sqrt{5} | + | d(S,r)-\lambda | + \\ &| d(i,j)-\frac{\lambda}{2}| + | d(i,l)-\frac{\lambda}{2}\times\sqrt{2} | + | d(i,m)-\frac{\lambda}{2}| + | d(i,n)-\frac{\lambda}{2}\times\sqrt{2} | + \\ &| d(i,p)-\frac{\lambda}{2}\times\sqrt{5} | + | d(i,q)-\lambda | + | d(i,r)-\frac{\lambda}{2}\times\sqrt{5} | + \\ &| d(j,l)-\frac{\lambda}{2}| + | d(j,m)-\frac{\lambda}{2}\times\sqrt{2} | + | d(j,n)-\frac{\lambda}{2}\times\sqrt{5} | + | d(j,p)-\lambda | + \\ &| d(j,q)-\frac{\lambda}{2}\times\sqrt{5} | + | d(j,r)-\lambda\times\sqrt{2} | + | d(l,m)-\frac{\lambda}{2}| + | d(l,n)-\lambda | + \\ &| d(l,p)-\frac{\lambda}{2}| + | d(l,q)-\frac{\lambda}{2}\times\sqrt{2} | + | d(l,r)-\frac{\lambda}{2}\times\sqrt{5} | + \\ &| d(m,n)-\frac{\lambda}{2}| + | d(m,p)-\frac{\lambda}{2}\times\sqrt{2} | + | d(m,q)-\frac{\lambda}{2}| + | d(m,r)-\frac{\lambda}{2}\times\sqrt{2} | + \\ &| d(n,p)-\frac{\lambda}{2}\times\sqrt{5} | + | d(n,q)-\frac{\lambda}{2}\times\sqrt{2} | + | d(n,r)-\frac{\lambda}{2}| + \\ &| d(p,q)-\frac{\lambda}{2}| + | d(p,r)-\lambda | + | d(q,r)-\frac{\lambda}{2}|\end{aligned} \tag{21}$$

Equation (19) applies only to the geometry of Figure 14 (a) (where the CH is at the South West corner of the array). For the other two geometries (CH in the center or CH in the base), the equations for minimization can be formed in a similar fashion.

The solution to our problem would be the nodes that minimize (19). Theoretically, the nodes that would provide this result would be positioned as in Figure 14 (a). This approach would examine all the possible node arrangements within the cluster, and the solution selected would be the one that best approximates the theoretical case.

2. Planar Array Formation using the Least Squares Line Fitting Approach

In this approach, first, a least squares fitted line is formed for the nodes positioned within the CH's communication distance (i.e., the cluster) and is an extension of the linear array construction using the least squares line fitting. Next, a set of lines (lines γ_1, γ_2, etc.), as needed, that are parallel to Line γ are obtained. These parallel lines, of course, are spaced $\lambda/2$ distance apart.

Figure 17 (a) shows the nodes within the cluster and the least squares fitted line γ as described in the previous section and the two parallel Lines γ_1 and γ_2 at a distance of $\lambda/2$ on both sides. The heavy dots (marked with black) are the ideal node locations, which are relative to the projection of the CH on Line γ and are exactly $\lambda/2$ distance apart.

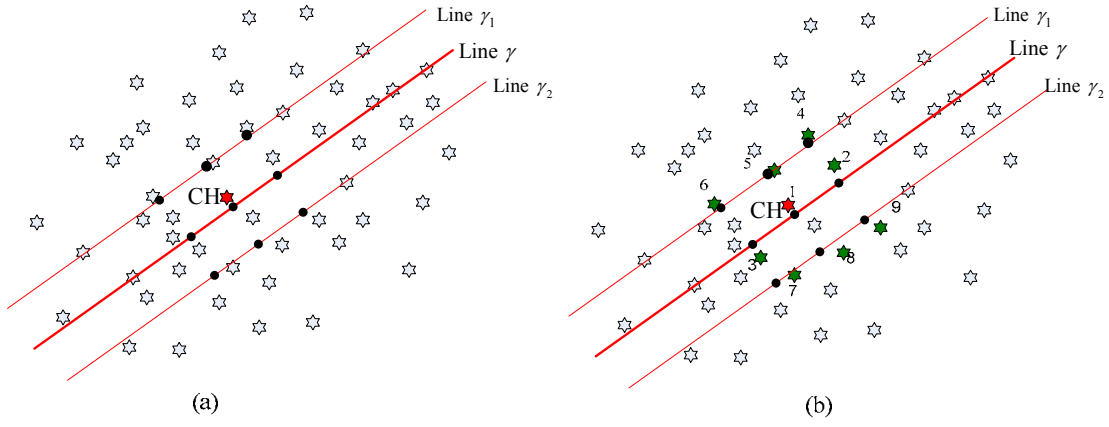

Figure 17. Nodes in Cluster to form a 3×3 Planar Array: (a) Least Squares Fitted Lines and Ideal Node Positions; (b) Selected Sensor Nodes that are Closest to the Ideal Locations

To form the desired planar array, we select the nodes that are positioned closest to the dots on Lines γ, γ_1 and γ_2. A possible solution is illustrated as nodes marked with green in Figure 17 (b).

Seeking another (possibly better) solution, we rotate the three parallel lines counter clockwise by an angle φ, which again is determined heuristically based on the node density in the cluster (larger angles for smaller node density and smaller angles for larger node density). The rotated lines and a possible solution are illustrated in Figure 18. Taking a careful look at Figure 18 (b), we observe that only one different node is selected when compared with the previous solution found in the Figure17 (b) above (the node in the North West corner, numbered as 6, is different).

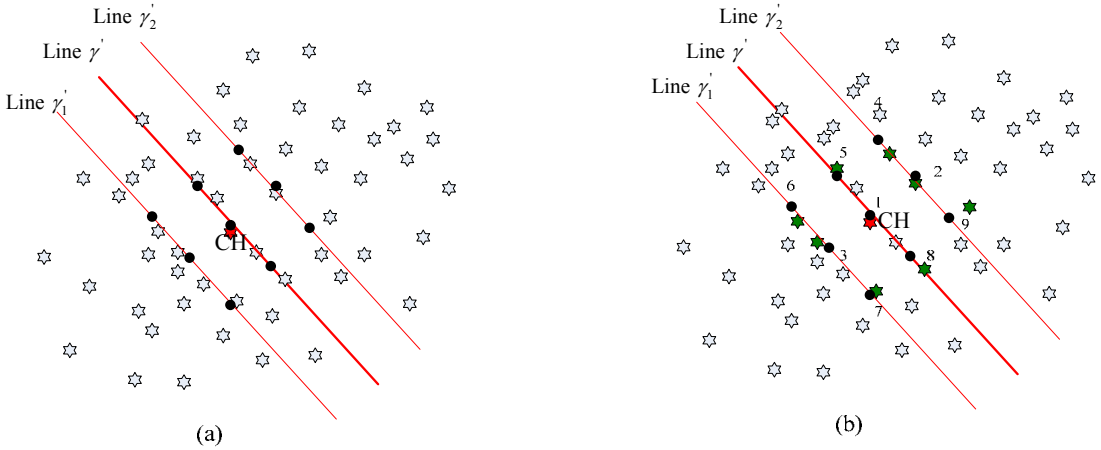

Figure 18. Fitted Lines Rotated by an Angle φ in the counter clockwise direction: (a) Least Squares Fitted Lines and Ideal Node Positions; (b) Selected Sensor Nodes that are Closest to the Ideal

After a heuristically-determined number of rotations, five in our simulations, the best solution with the minimum error, in terms of distance from the edges, is selected. These nodes will then be assigned to coordinate their transmissions to perform beamforming and transmit the data to a distributed analysis center through the overhead UAV.

C. SUMMARY

In this chapter, we proposed methods of forming linear and planar arrays in a sensor field without the aid of GPS or other location knowledge. In the linear array

formation, we presented three methods: iterative, concurrent and line fitting approach. Expanding the concurrent and the line fitting approaches, we proposed two approaches for planar array formation.

In the next chapter, we present the simulation results to evaluate the performance of the array formation techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PERFORMANCE ANALYSIS AND SIMULATION RESULTS

This chapter presents and evaluates the performance of the proposed approaches for constructing an array in a WSN field. First, we present a set of performance evaluation metrics used in the simulations. Next, the simulation model is discussed and, finally, the simulations results for the proposed methods of Chapter III for the construction of linear and planar arrays are presented. For the linear array, the results presented are from the iterative and the line fitting approaches. For the planar array, the results are presented for the line fitting approach.

A. PERFORMANCE EVALUATION METRICS

The results from the simulations must be analyzed to determine the effectiveness of the approach in forming a linear array. The array that is constructed in the field must be compared with an array that is perfectly linear with inter-node spacing of $\lambda/2$. The nodes that are in perfect linear positions and have interspacing distance of $\lambda/2$ would give the highest gain.

To evaluate the performance of each method, we introduce a set of metrics used for calculating the errors of the array formed. These metrics will be used to evaluate the performance based on the simulation results. The three proposed error metrics are defined as follows.

1. Perpendicular Distance, ε_p: a measurement of the perpendicular distance of each node of the constructed array from the line with ideal node locations.

2. Inter-node Spacing Error, ε_s: a measurement of the difference in inter-node spacing along the line between the orthogonal projection of the nodes and the ideal positions at multiples of $\lambda/2$.

3. Total Error, ε_t: a measurement of the Euclidean distance between the nodes and the ideal positions on the line representing the perfect array

The remainder of this section describes the details of the calculation of these metrics. All three metrics are used to evaluate the results from Matlab simulations for

linear array using the iterative and the line fitting methods. Before attempting to determine the metrics, we need to determine a reference line. In the case of the least squares line fitting approach, the reference line is the fitted line. For the iterative approach, a line needs to be determined for computing the metrics.

1. Determination of the Reference Line

The reference line is obtained using the least squares regression approach for the selected nodes. For example, Figure 20 illustrates a five-node array with a reference line. The nodes of the constructed array introduce errors in both dimensions. The ideal positions are assumed to lie on the line with inter-node distances of $\lambda/2$.

Let S represent the set of ideal node positions along the reference line:

$$S = \{S_i\} = \{(x_i, y_i)\} \tag{22}$$

and let S' represent the actual node positions around the line defined as

$$S' = \{S_i'\} = \{(x_i', y_i')\} \tag{23}$$

where all positions are relative.

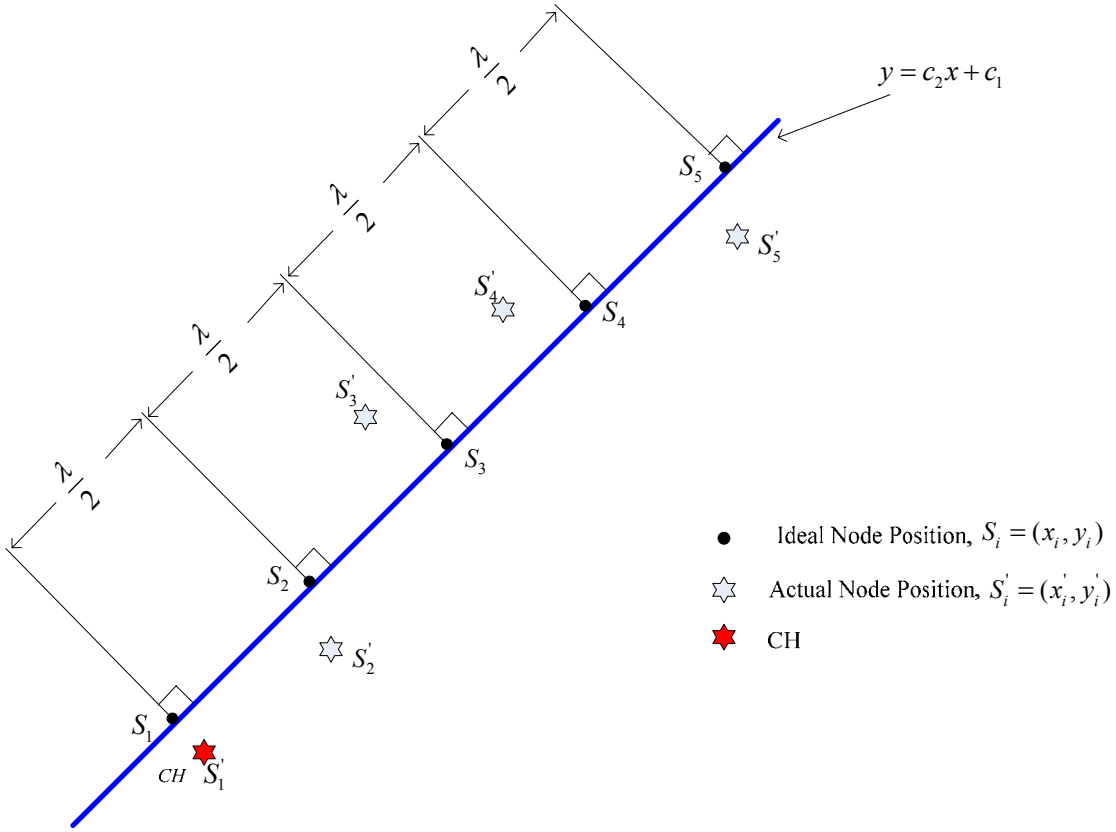

Figure 19. Sensor Nodes and the Reference Line

Given the node position data (x_i', y_i') for n nodes, a regression line using least squares is obtained to represent the reference line. The Matlab function *regress* (X, y) is used for this purpose, where y is a vector of the *y*-coordinate values

$$y = \begin{bmatrix} y_1' \\ y_2' \\ . \\ . \\ y_n' \end{bmatrix} \tag{24}$$

and X is an $n \times 2$ matrix defined as

$$X = \begin{bmatrix} 1 & x_1' \\ 1 & x_2' \\ . & \\ . & \\ 1 & x_n' \end{bmatrix} \tag{25}$$

and n is the number of nodes in the array.

The function solves the equation

$$y = Xc \tag{26}$$

where c is a 2×1 vector

$$c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \tag{27}$$

with c_2 representing the slope of the line and c_1 the offset from the origin. The reference line can be expressed as

$$y = c_2 x + c_1 . \tag{28}$$

Without loss of generality, we can assume that the CH is located at the origin and that $c_1 = 0$.

2. Perpendicular Distance Error Metric

The first error calculated, which gives an estimate of the achieved approximation in linearity of the nodes in simulation, is the perpendicular distance of the nodes from the reference line. Figure 21 illustrates the perpendicular distances.

The slope of the line perpendicular to the reference line is $-\frac{1}{c_2}$. For a given node i, the intersection of the perpendicular line and the reference (x_{p_i}, y_{p_i}) is obtained by solving the two line equations

$$y = c_2 x + c_1 \tag{29}$$

$$y = -\frac{1}{c_2} x + c_3 \tag{30}$$

where c_3 is the offset of the perpendicular line associated with node i. The set of these intersections is $S_p = \{S_{p_i}\} = \{(x_{p_i}, y_{p_i})\}$. The perpendicular distance is then given by [16]

$$\varepsilon_{p_i} = \sqrt{(x_i^{'} - x_{p_i})^2 + (y_i^{'} - y_{p_i})^2} \ . \tag{31}$$

The average perpendicular error given by

$$\varepsilon_p = \frac{1}{n}\sum_{n=1}^{n} \varepsilon_{p_i} \tag{32}$$

is used as the metric to represent the perpendicular distance error. Even though this approach provides a metric for comparing the different arrays, it does not give us any information regarding the inter-node distances. To evaluate the inter-node spacing, we introduce the second metric in the next section.

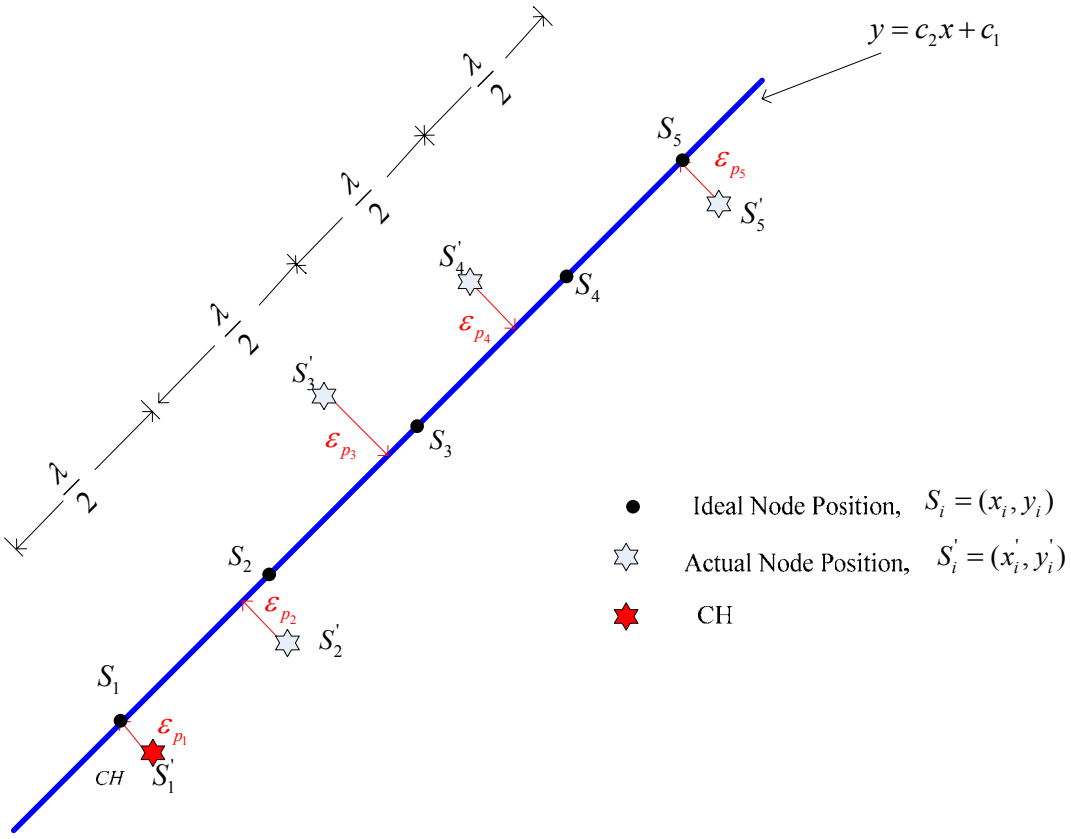

Figure 20. Perpendicular Distance Error Measurement

3. Inter node Spacing Error metric

The inter-node spacing between the successive ideal node positions along the reference line is $\lambda/2$. Figure 22 illustrates the spacing among the ideal nodes as well as the spacing error (along the line) for the actual nodes.

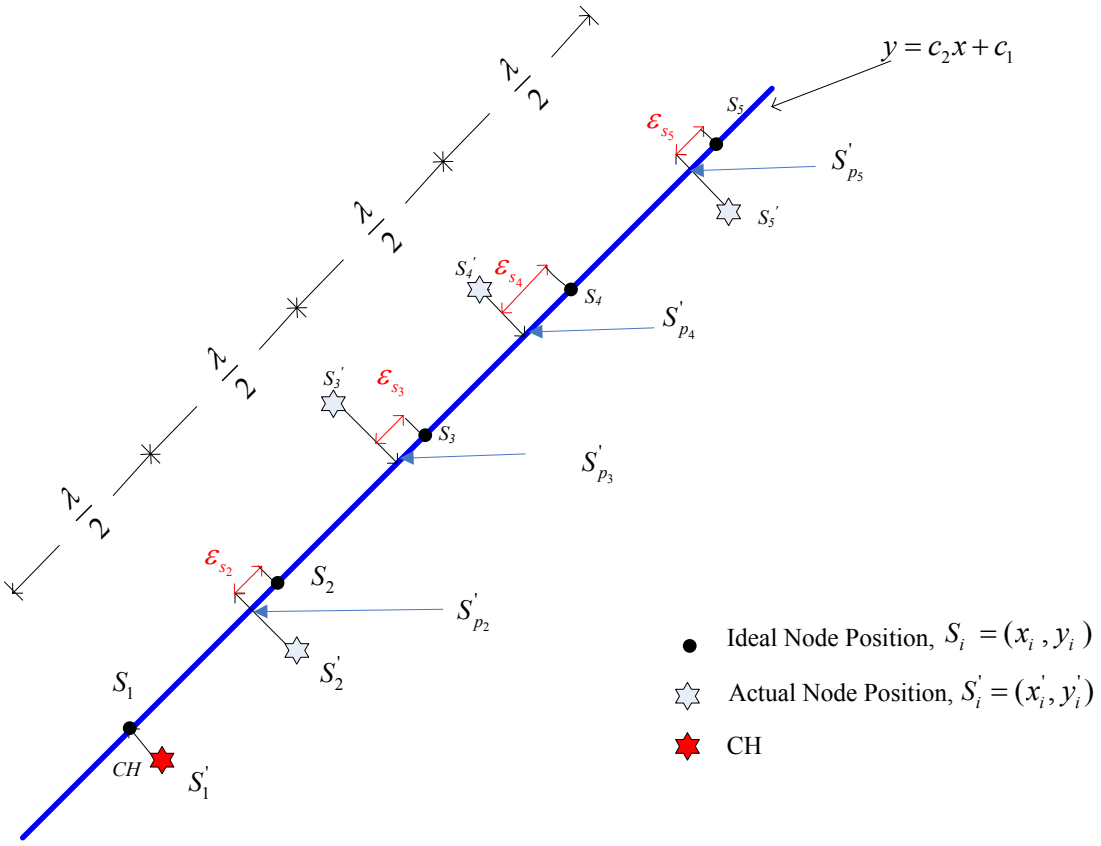

Figure 21. Internode Spacing Error between Ideal Node Positions and Perpedicular Projections of the Actual Nodes along the Reference Line.

The inter-node spacing error for the i^{th} node is given by

$$\varepsilon_{s_i} = \sqrt{(x_i - x_{p_i})^2 + (y_i - y_{p_i})^2} \tag{33}$$

and the internode spacing error metric represented by the average is given by

$$\varepsilon_s = \frac{1}{n}\sum_{n=1}^{n} \varepsilon_{s_i} \ . \tag{34}$$

The frequency and density that we use effect the inter-node spacing error. It cannot be used to compare simulations where the frequency of operation is different.

4. Total Distance Error Metric

The total error for the i^{th} node is calculated as

$$\varepsilon_{t_i} = \sqrt{(x_i - x_i^{'})^2 + (y_i - y_i^{'})^2} \tag{35}$$

where (x_i, y_i) represents the ideal position of the i^{th} node and $(x_i^{'}, y_i^{'})$ represents the actual position. Figure 23 illustrates the individual total errors. The total error metric ε_t is the average of all the individual errors given by

$$\varepsilon_t = \frac{1}{n}\sum_{i=1}^{n}\varepsilon_{t_i} \; . \tag{36}$$

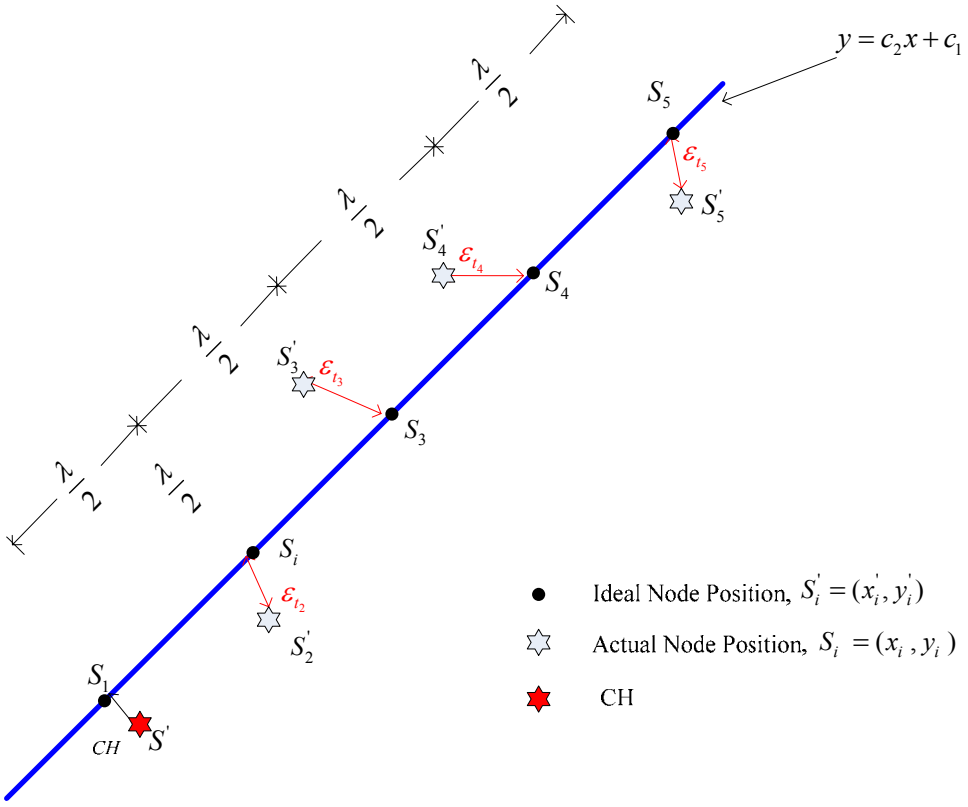

Figure 22. Total Error Calculation

5. Planar Distance Error

In the case of planar antenna array construction, the calculation of error in the resulting array is performed by calculating the Euclidean distance between the ideal position and the actual position of the node. The ideal positions of the nodes are on a square grid as described in Chapter III.

Let S' be the set of position of all nodes in the array

$$S' = \{S'_{ij}\} = \{(x'_i, y'_j)\}, \quad i = 1, 2, ..., n \text{ and } j=1, 2, ..., n$$

and let S be the set of the ideal node positions (on the grid)

$$S = \{S_{ij}\} = \{(x_i, y_i)\}, \quad i = 1, 2, ..., n \text{ and } j=1, 2, ..., n$$

The Euclidean distances between a node and idel position is given by

$$\varepsilon_{ij} = \sqrt{(x_i - x'_i)^2 + (y_j - y'_j)^2} \tag{37}$$

The Euclidean distance error for a 3×3 planar array is illustrated in Figure 23 as between a node and the corresponding ideal position. The CH is aligned to the center of the array and the ideal position and errors of the remaining eight nodes are calculated.

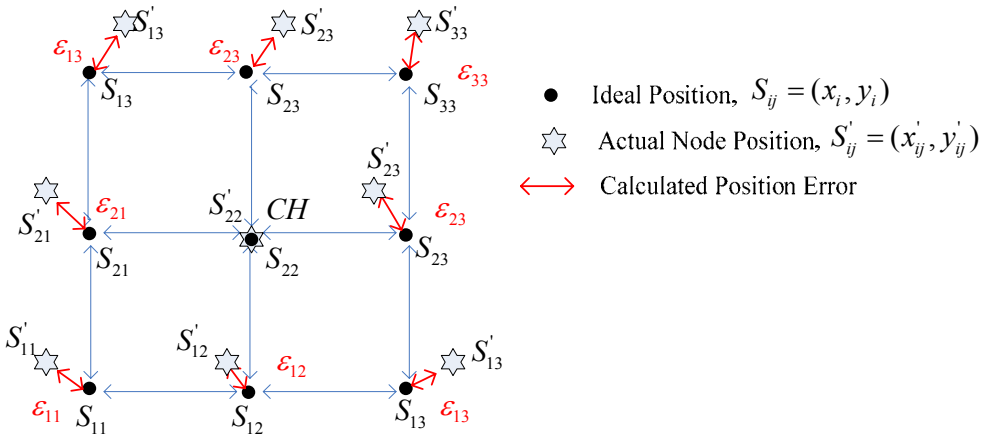

Figure 23. Calculated Position Error for a Planar Antenna Array

The total error ε_t is the average of all the individual errors given by

$$\varepsilon_t = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \varepsilon_{ij} \tag{38}$$

where the error corresponding to the CH (in Figure 24, ε_{22}) is zero.

B. SIMULATION MODEL

A simulation model was developed in MATLAB code to study and compare the linear and planar array formation for multiple scenarios using different communication frequencies, communication range and node densities. All simulations were performed in an area of 10×10 meters^2 with communication ranges of 2 and 4 meters for both linear and planar array formation. The linear array formation was simulated for 7 and 9 nodes while the planar array was 3×3 and 4×4. Node density ranged from 100 to 1500 and frequencies of 300 MHz and 900 MHz were used in the simulations.

1. Inter-node Distance and Density in the Field

The $\lambda / 2$ inter-node spacing distance we are trying to achieve between adjacent nodes depends only on the communication frequency of the nodes. We have

$$d = \frac{\lambda}{2} = \frac{c}{2f} \tag{39}$$

where d is the inter-node spacing, c is the speed of light (3×10^8 m/s), f is the communicating frequency of the nodes (used to perform beamforming) and λ is the effective wavelength. The appropriate inter-node spacing decreases as the communication frequency of the nodes increases. The higher the frequency used for inter-node communication purposes, and thus for beamforming purposes, the lower the inter-node distance must be. This results in the need for higher density in the specified subgroup, which is assigned to form the desired linear or planar array.

The density depends on the size of the field and the total number of the nodes that we are capable of deploying in the area. Recall from (1) that the density of the nodes would be higher if the number of nodes is relatively high and the sensor field is small. The opposite would occur if the number of nodes is relatively low and the field is large.

The size of the field used to perform the simulations was 10×10 meters^2, and communication range of the nodes was chosen to be 2 meters for the 7-node array and 4 meters for the 9-node array case. This results in a field percentage of coverage for each

node of $\pi r^2 / A^2 = \pi 2^2 / 10^2 = 0.1256$ or 12.56% of the whole sensor network area for the 2-meter communication distance and $\pi r^2 / A^2 = \pi 4^2 / 10^2 = 0.5024$ or 50.24% for the 4-meter case.

The number of nodes used in the simulation ranged from 100 to 1500 in steps of 100 for each simulation. A uniform distribution was assumed within the field. As the number of nodes in the area increases, the density also increases. This resultant node density within the cluster is shown in Table 1. This is the number of nodes in the transmission radius of the CH (i.e., cluster). As we can see in Table 1, using (1), the number of nodes within the communication radius of the CH varies from approximately 12 to 180 nodes for communication radius 2 meters and 50 to 753 for communication radius 4 meters.

Number of Nodes ($\times 10^2$)	**Average cluster size (2m comm. radius)**	**Average cluster size (4m comm. radius)**
1	12.56	50.24
2	25.12	100.48
3	37.68	150.72
4	50.24	200.96
5	62.80	251.2
6	75.36	301.44
7	87.92	351.68
8	100.48	401.92
9	113.04	452.16
10	125.60	502.4
11	138.16	552.64
12	150.72	602.88
13	163.28	653.12
14	175.84	703.36
15	188.40	753.6

Table 1. Density in the Sensor Field and Number of Nodes in a Cluster in a $10 \times 10\,\mathrm{m}^2$ field

Table 1 is graphically represented in Figure 26, where we can see how the number of the nodes within a cluster increases with the total number of nodes in the sensor field of 10×10 meters^2 for CH communication radii of 2 and 4 meters.

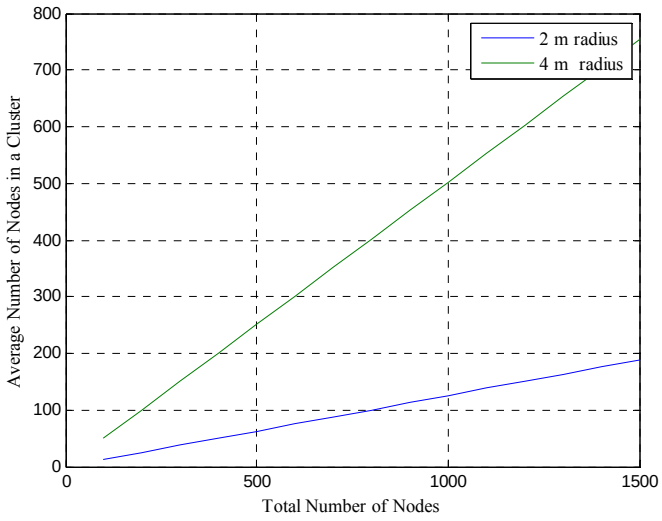

Figure 24. Number of Nodes within a Cluster for 2 m and 4 m Communication Distance for a $10 \times 10\,\text{m}^2$ sensor field

2. Simulation Model

A simulation model was developed in MATLAB to evaluate the proposed linear and planar antenna array formation methods in a wireless sensor field. Figure 25 presents the flowchart of the simulation.

The flowchart in Figure 25 has two main parts. The first part is the construction and evaluation of a linear antenna array, and the second separate part is the construction and evaluation of a planar antenna array. Both parts start with the construction of the simulation sensor field, the uniform deployment of the sensor nodes and, finally, the selection of the CH.

Initially, the sensor field is formatted and a desired number of nodes is distributed uniformly into it. According to the selected communication distance of the nodes, the links among them are formed. Two methods are performed for the selection of the linear array: the iterative approach and the least squares line fitting approach. The iterative

approach antenna array initially forms a 3-node linear array and then, according to the desired total number of nodes, each additional element of the array is added up to the total number of nodes.

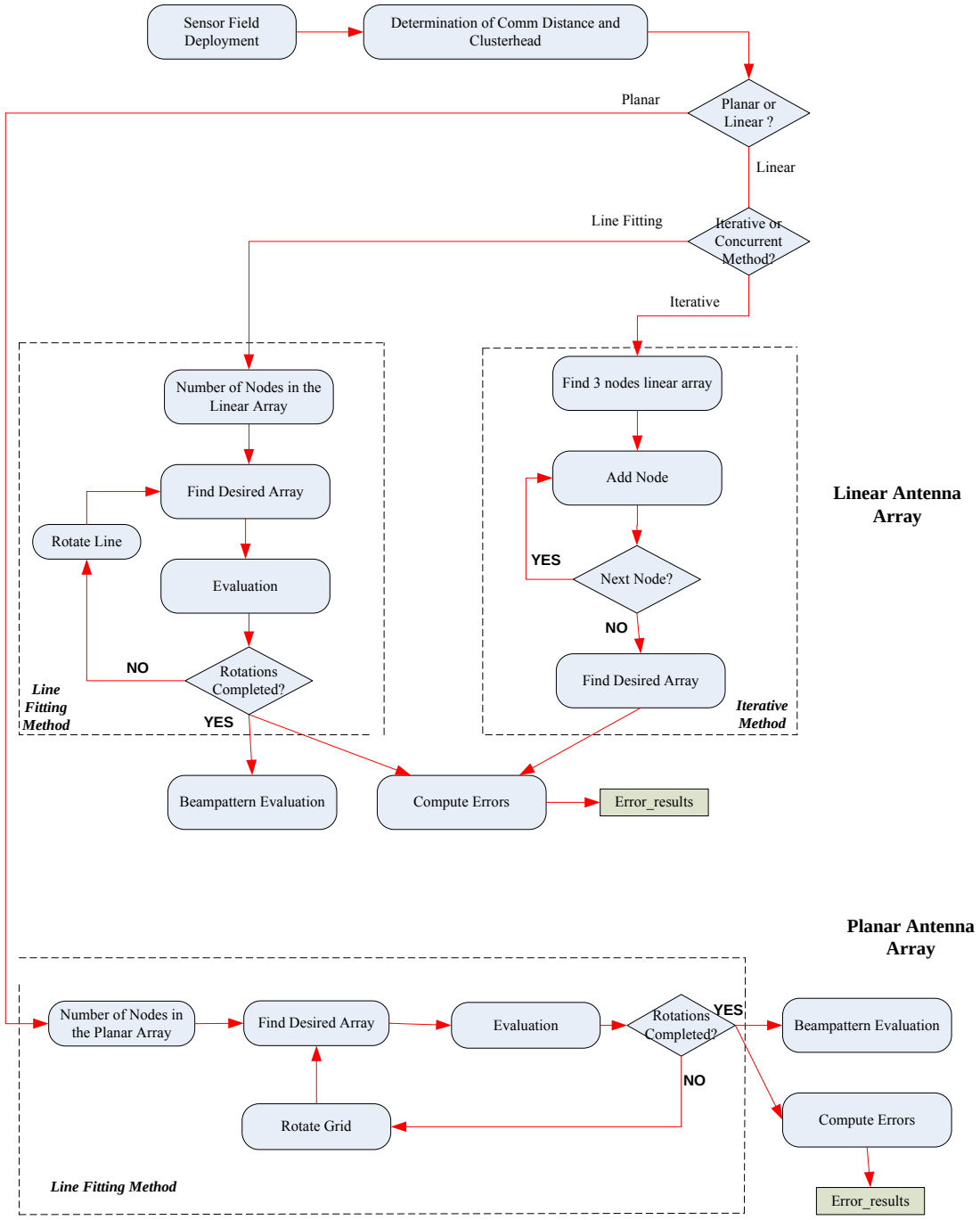

Figure 25. Simulation Model for the Construction and Evaluation of Linear and Planar Antenna Array

For the line fitting method, the regression line based on least squares is determined within the cluster and then the ideal node positions are marked. The algorithm evaluates each solution, and after a number of predetermined rotations, the solution that provides the best total error metrics is provided. A comparison is performed between the two solutions based on the metrics described earlier.

For the construction of a planar array, the construction of the sensor field, the selection of CH and the selection of the connectivity of the nodes is performed as in the linear antenna array. Next, the least squares line fitting provides a number of solutions, based on the number of rotations selected. Finally, the solution that provides the best total error metric is selected.

The MATLAB functions developed for this simulation are included in Appendix A.

C. LINEAR ARRAY RESULTS

Simulations were performed for both iterative linear array formation and the line fitting method for a variety of inputs (i.e., number of nodes in the field and frequency of operation). 1000 Monte Carlo runs for each case were performed and the average metrics were calculated. The area of the sensor network was 10×10 meters^2, and the communication range of each node was 2 meters. Two frequencies of operation were used: 300 MHz and 900 MHz. The simulation input data are summarized in Table 2.

For each one of the four simulations, the number of nodes ranged from 100 to 1500 and the resulting density ranged from 1 node / m^2 to 15 nodes / m^2. For each density, a Monte Carlo simulation of 1000 runs was performed and the mean of each of the three errors was calculated, using the metrics described earlier.

Simulation Input Data				
Frequency (MHz)	300	300	900	900
$\lambda/2$ (m)	0.500	0.500	0.167	0.167
Number of Runs (Monte Carlo)	1000	1000	1000	1000
Sensor Field Dimensions (m)	10×10	10×10	10×10	10×10
CH Communication Radius (m)	4	4	2	2
Linear Array Size (# of nodes)	7	9	7	9

Table 2. Simulation Input Data for Linear Array Formation

1. Seven-node Array at 300 MHz

The first simulation for linear array formation was performed at 300 MHz with an array size of 7 nodes for various densities in an area of 10×10 meters^2. The perpendicular distance, inter-node spacing and total error are plotted in Figures 26 to 28. The green line represents the line fitting approach and the red line represents the iterative approach.

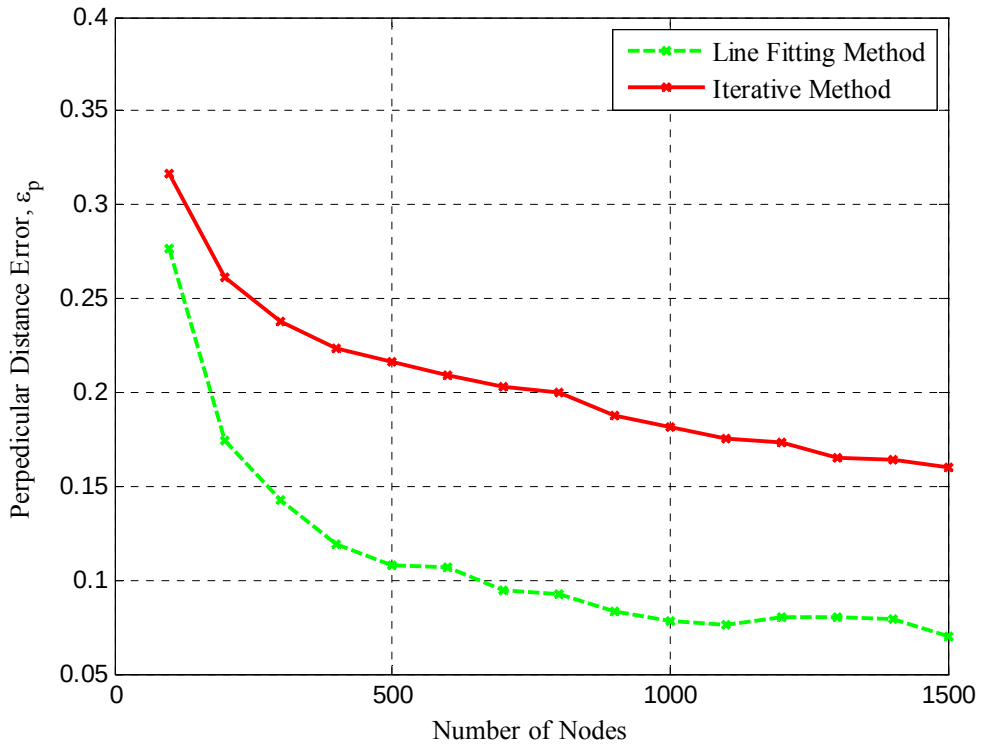

Figure 26. Average Perpendicular Distance Error ε_p for 300 MHz and 7 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

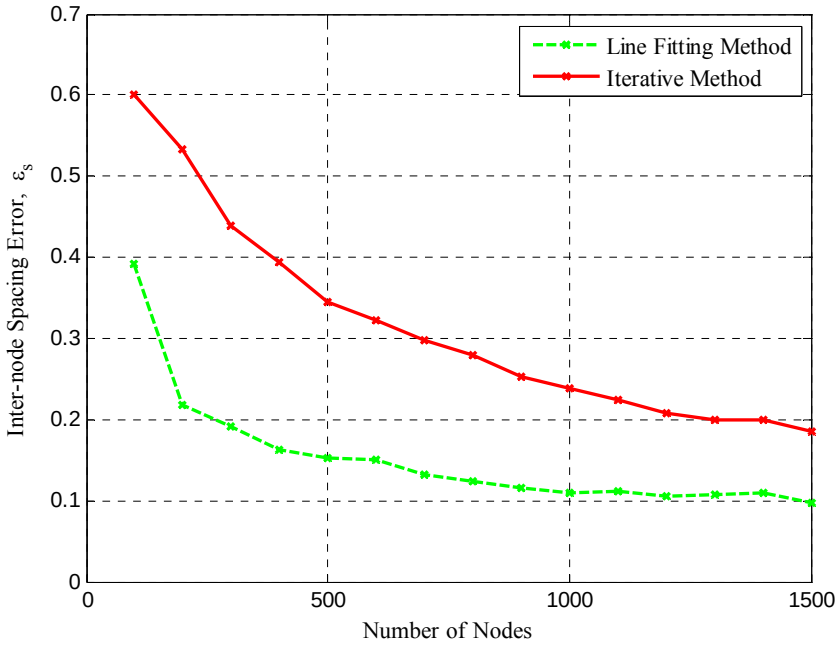

Figure 27. Average Inter-node Spacing Error ε_s for 300 MHz and 7 Linear Nodes Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

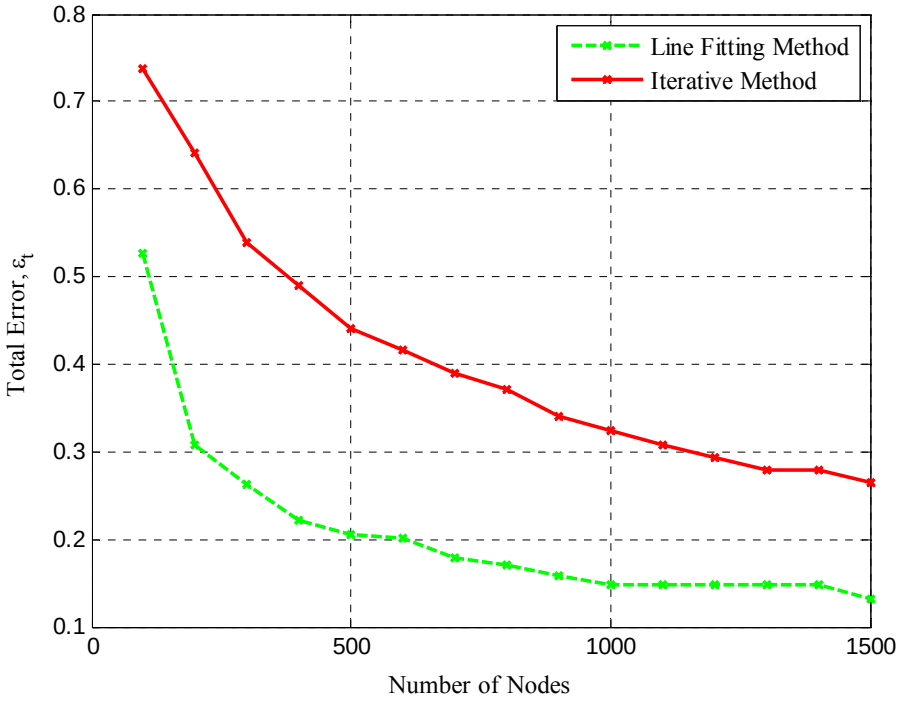

Figure 28. Average Total Error ε_t for 300 MHz and 7 Nodes Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

As we can see, all three error metrics are significantly lower for the line fitting approach compared with the iterative approach. Both methods show improvement in linearity with increase in density.

The error metrics for both methods are greater for lower densities with the greatest at 100 nodes in the 10×10 m^2 field. The minimum errors are obtained for the highest density of 1500 nodes in same field. The line fitting approach remains almost steady after a density of 600 nodes, as compared with the iterative approach for which the errors continue to decrease for densities higher than 600 nodes, and appears to stabilize only after about 1200 nodes in the field.

The average beampattern for multiple node densities can be plotted to show the effectiveness of the line fitting method in the construction of a linear array [17]. Figures 29 to 31 present the average beampattern for field densities of 300, 600 and 1000 nodes.

The blue line in Figures 29 to 31 represents the ideal array beampattern when the nodes are in the ideal positions (i.e., inter-node distance of $\lambda / 2$ with no linearity error). The red line is the result of the average beampattern for the simulations performed (i.e., actual positions of the nodes by using the line fitting method for constructing the 7-node linear array).

Comparing the beampatterns for the three different densities, we see that for 300 nodes the main gain lobe is reduced and the side lobe gain is increased. For 600 nodes, the main lobe gain is close to ideal and the side lobe gain has improved. The 1000-node beampattern shows further improved main lobe as compared with the 600-node case and the nulls show significant improvement. At the density of 600 nodes/m^2 the results show the best sidelobe level relative to the main beam.

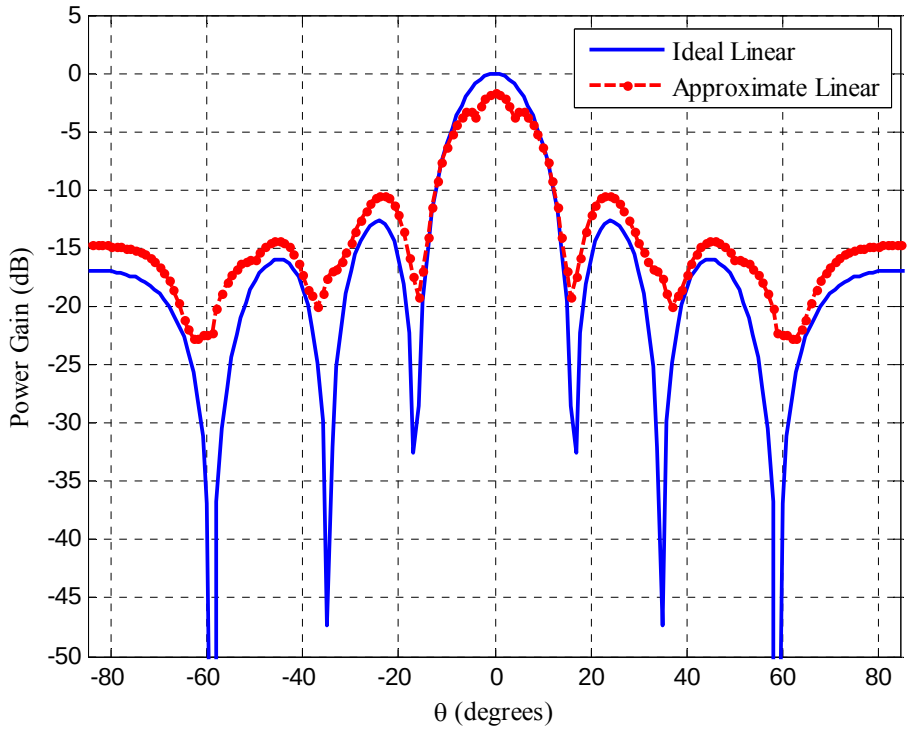

Figure 29. Average Beampattern for Uniform and Approximately Linear 7 Elements Array at 300 MHz for 300 Nodes for 1000 Monte Carlo Runs

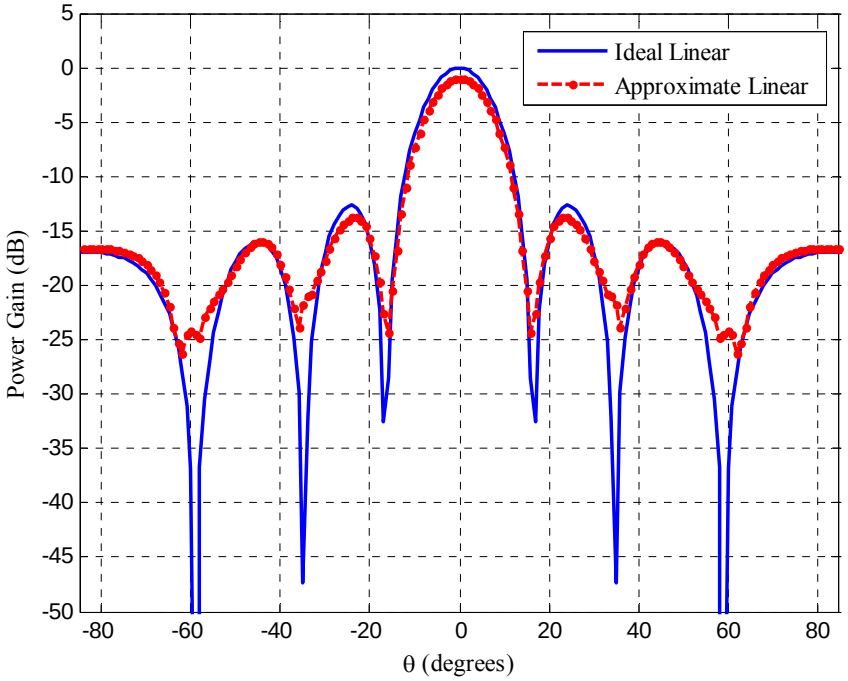

Figure 30. Average Beampattern for Uniform and Approximately Linear 7 Elements Array at 300 MHz for 600 Nodes for 1000 Monte Carlo Runs

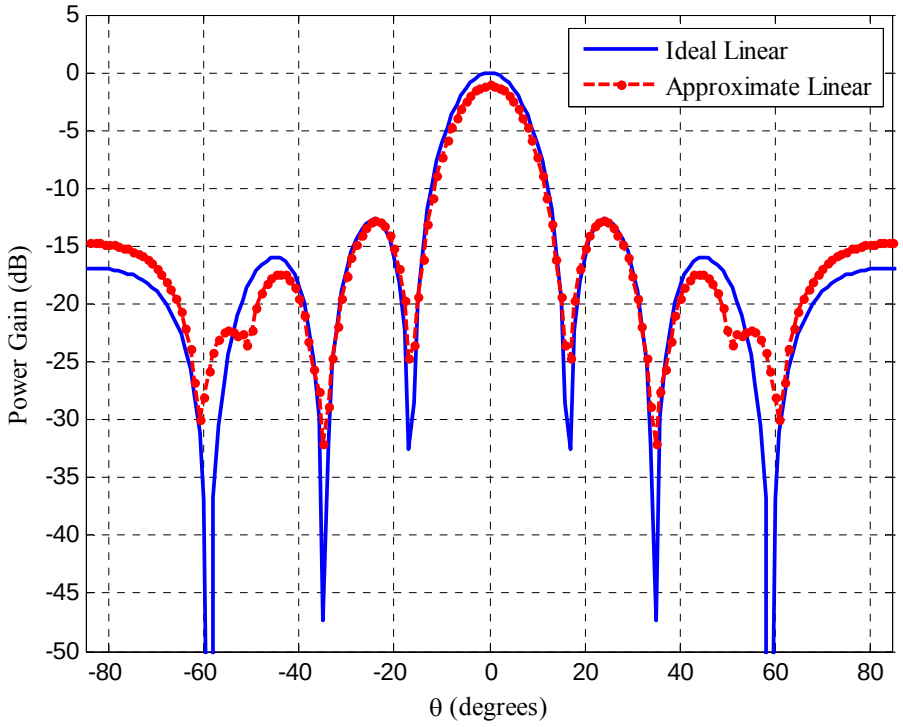

Figure 31. Average Beampattern for Uniform and Approximately Linear 7 Elements Array at 300 MHz for 1000 Nodes for 1000 Monte Carlo Runs

2. Nine-node Array at 300 MHz

The second simulation was performed at 300 MHz for a 9-node linear array. Using the MATLAB function *Linear_line_fitting_comparison_iterative.m,* we construct an array of 9 nodes for various densities in an area of 10×10 meters^2. The perpendicular distance, inter-node spacing, and total error metrics are plotted in Figures 32 to 34. The green line represents the line fitting approach and the red line the iterative approach.

For the iterative approach, the perpendicular distance error ε_p is seen to increase compared to the 7-node simulation at the same densities (see Figure 32).

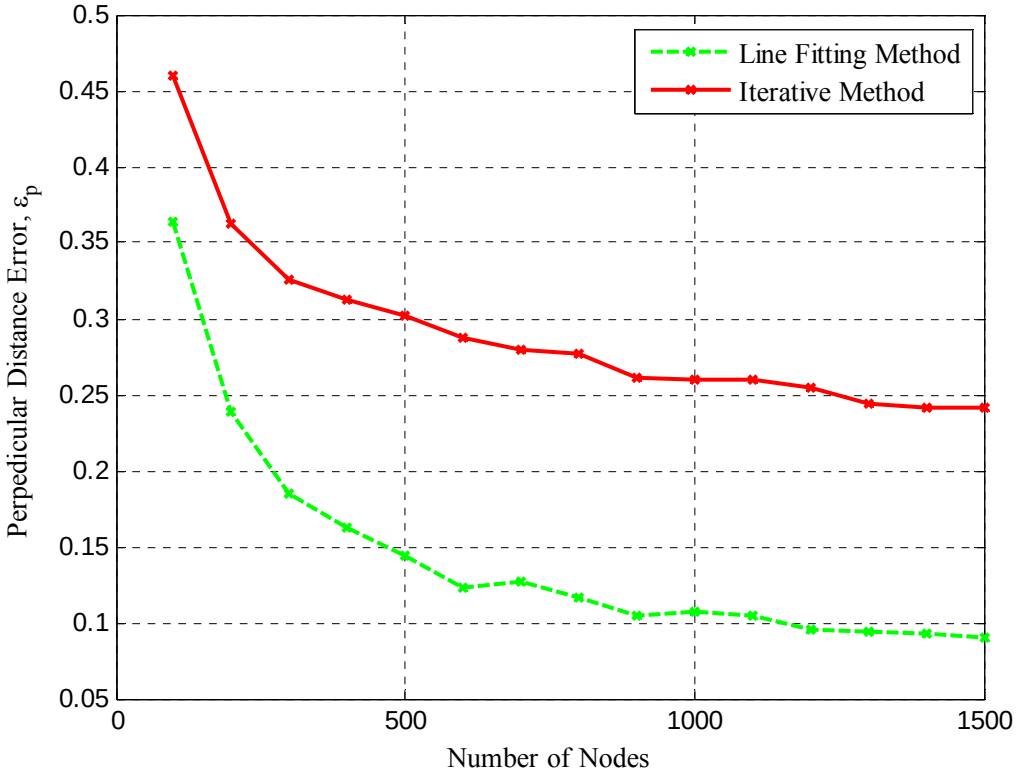

Figure 32. Average Perpendicular Distance Error ε_p for 300 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

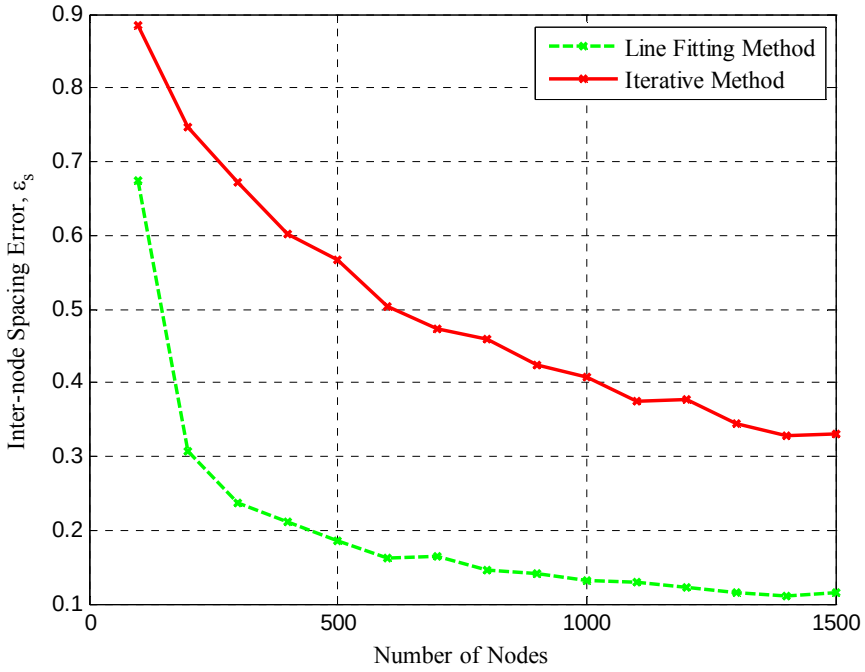

Figure 33. Average Inter-node Spacing Error ε_s for 300 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

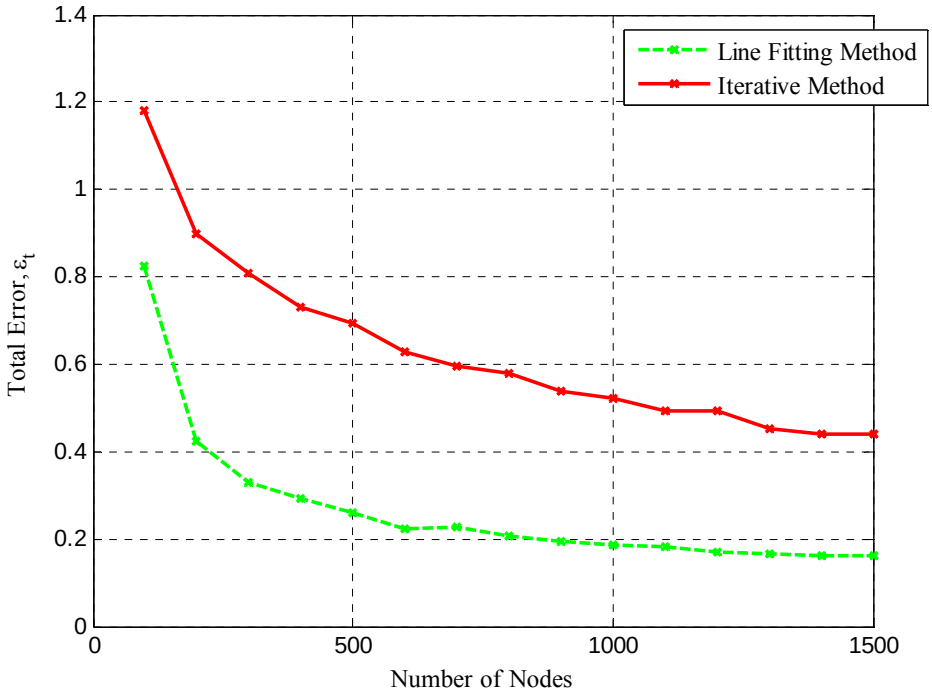

Figure 34. Average Total Error ε_t for 300 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

The inter-node spacing error for the line fitting approach for both the 7- and 9-node arrays (see Figures 27 and 33) is seen to be in same range between 0.1 and 0.2 meters. The difference in the perpendicular and inter-node spacing error for both methods is also reflected in the total error for both simulations (see Figures 28 and 34) with the iterative approach method yielding in larger errors.

Figure 35 represents the average beampattern for the line fitting method (red line) at a density of 600 nodes. The 600-node density is selected because it is the point where the total error begins to stabilize. As we can see, the beampattern approaches the ideal case in the main lobe with a slight increase in the side lobes. The average beampattern improves for higher node densities.

Comparing the results of both error metrics and beampatterns, we conclude that the performance improves with the density and that the line fitting approach provides better results in all cases when compared with the iterative method.

Additionally, the comparison between the 7- and 9-node simulations shows that a slight increase in error metrics for the 9-node case, but the average beampattern for the 9-node case is better than that of the 7-node array.

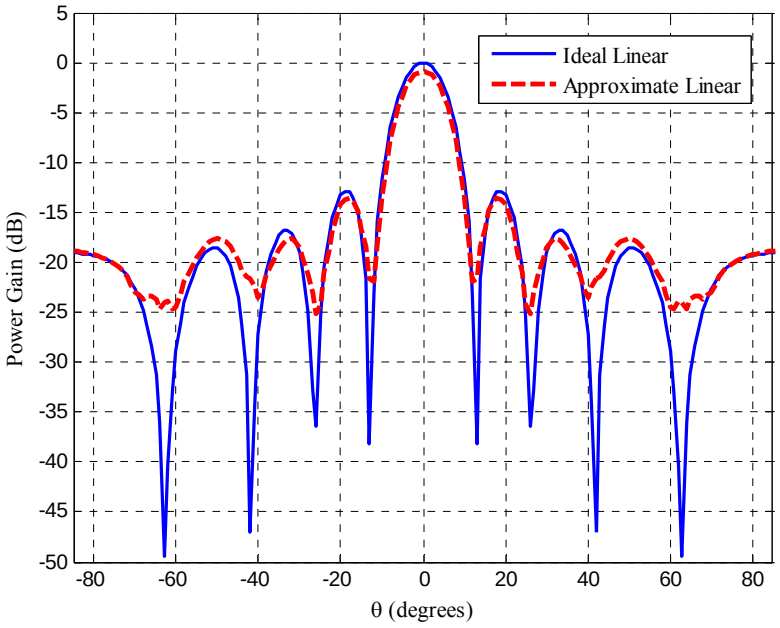

Figure 35. Average Beampattern for Uniform and Approximately Linear 9 Elements Array at 300 MHz for 600 Nodes for 1000 Monte Carlo Runs

3. Seven-node Array at 900 MHz

The third linear array formation simulation was performed at a higher frequency of 900 MHz for a 7-node case. We construct an array of 7 nodes for various densities in an area of 10×10 m^2. The perpendicular distance, inter-node spacing and total error are plotted in Figures 36 to 38. The green line represents the line fitting approach and the red line the iterative approach.

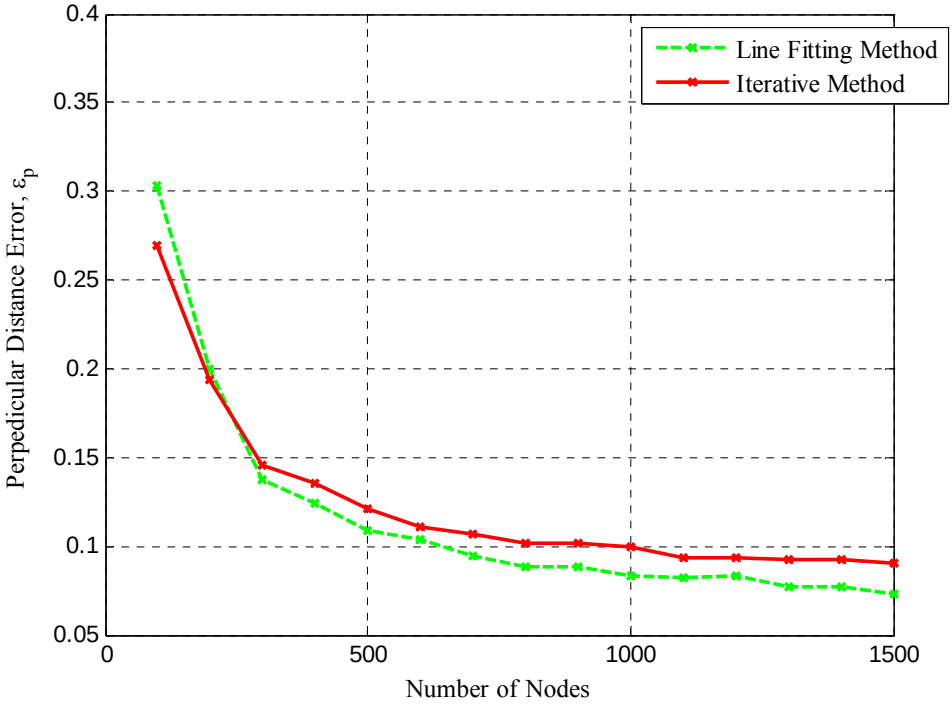

Figure 36. Average Perpendicular Distance Error ε_p for 900 MHz and 7 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

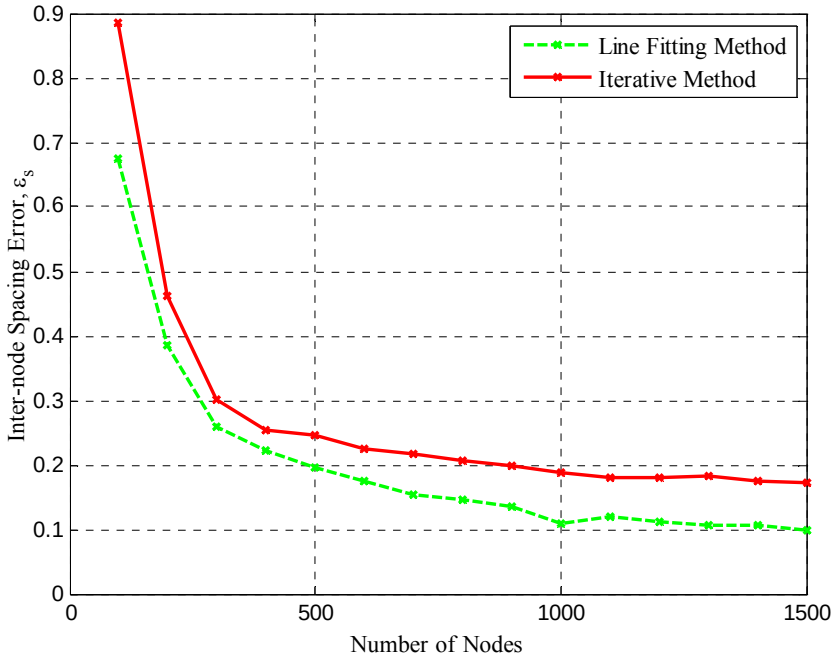

Figure 37. Average Inter-node Spacing ε_s Error for 900 MHz and 7 Nodes Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

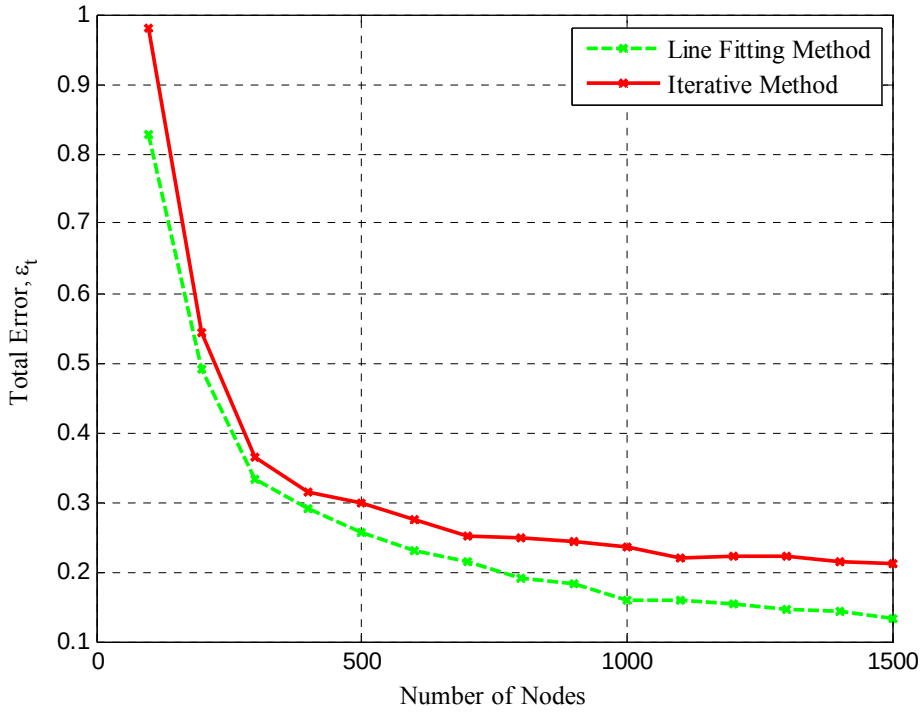

Figure 38. Average Total Error ε_t for 900 MHz and 7 Nodes Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

Figure 39 presents the average beampattern for the line fitting method (red line). The 600-node density is once again selected because from that point on the total error is nearly stabilized (below 0.23 meters). As we can see, the beampattern approaches the ideal case in the main lobe with an increase in the second and third side lobes. As in previous cases, the average beampattern approaches the ideal pattern for higher node densities in the field.

Summarizing the results of this simulation, we conclude that performance improves with density, but the line fitting method shows less improvement over the iterative method than in the previous cases. Additionally, the beampattern shows that for increased frequency and the same number of nodes, the performance degrades since the required inter-node spacing is reduced.

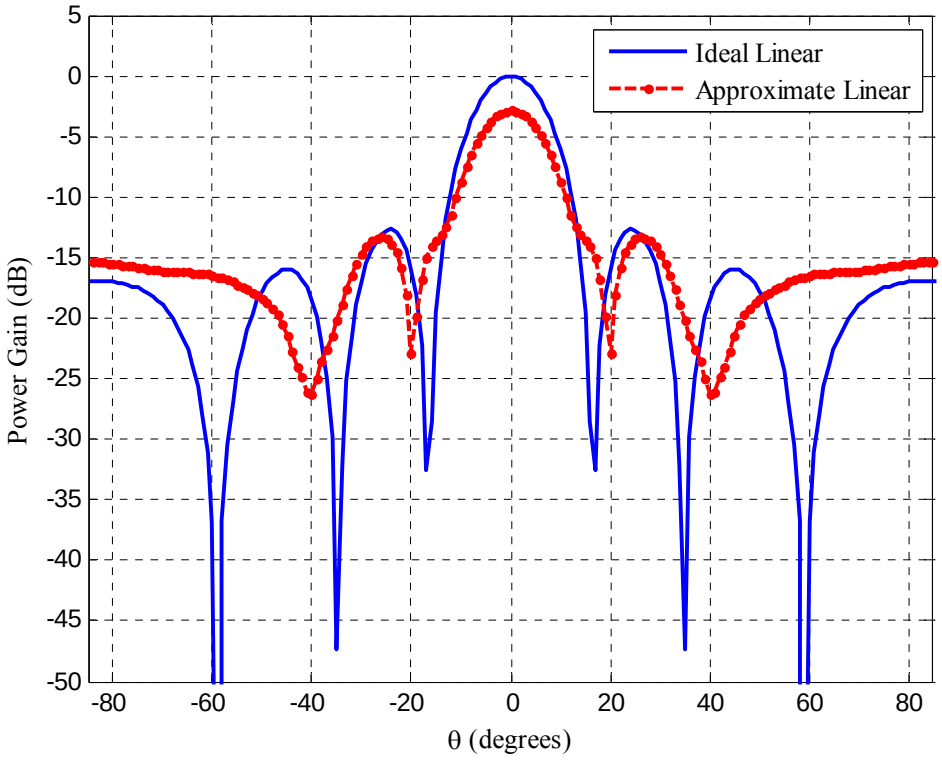

Figure 39. Average Beampattern for Uniform and Approximately Linear 7 Elements Array at 900 MHz for 600 Nodes for 1000 Monte Carlo Runs

4. Nine-node Array at 900 MHz

The final linear array formation simulation is performed at 900 MHz for 9 nodes for various node densities in a field of 10×10 meters^2. The perpendicular distance, inter-node spacing and total error are plotted in Figures 40 to 42. The green line represents the line fitting approach and the red line the iterative approach.

In this case, the performance increases with the density for both methods. The line fitting method maintains the same performance while the iterative solution does not improve when compared with the 7 nodes case for the same frequency. From the beampattern, we conclude that for the same frequency, more nodes in the array improve the performance.

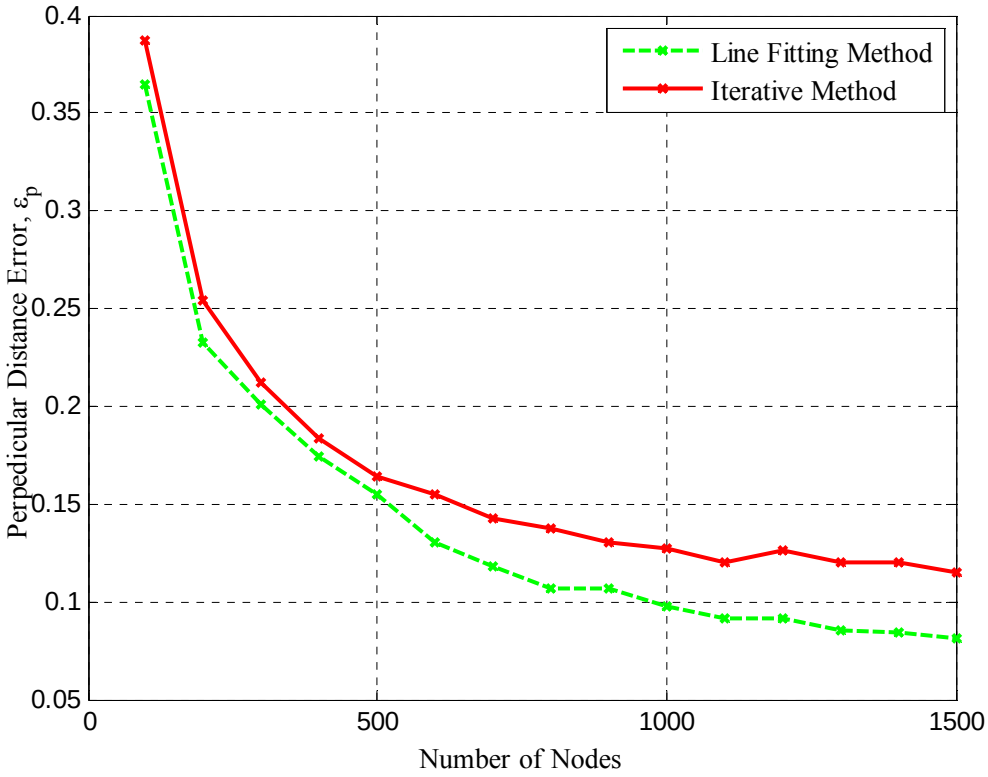

Figure 40. Average Perpendicular Distance Error ε_p for 900 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

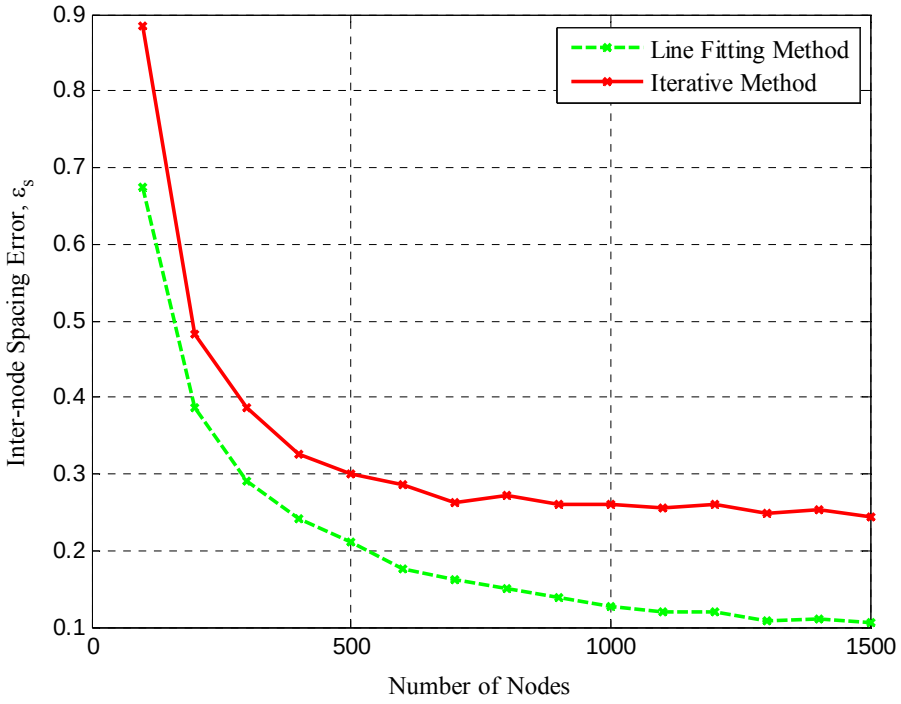

Figure 41. Average Inter-node Spacing Error ε_s for 900 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

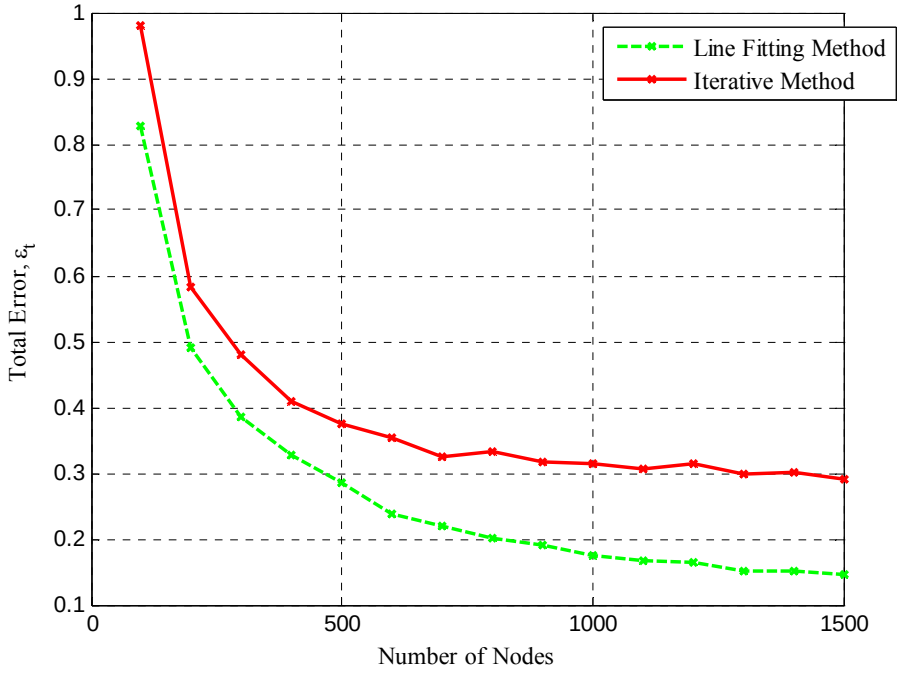

Figure 42. Average Total Error ε_t for 900 MHz and 9 Nodes Linear Antenna Array for 1000 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

Figure 43 shows the average beampattern for the line fitting method (red line). As in previous studies, the 600-node density is selected and we can see that the beampattern has improved compared to the 7-node case (see Figure 43). For the same frequency, the performance has improved as more nodes are used in the array.

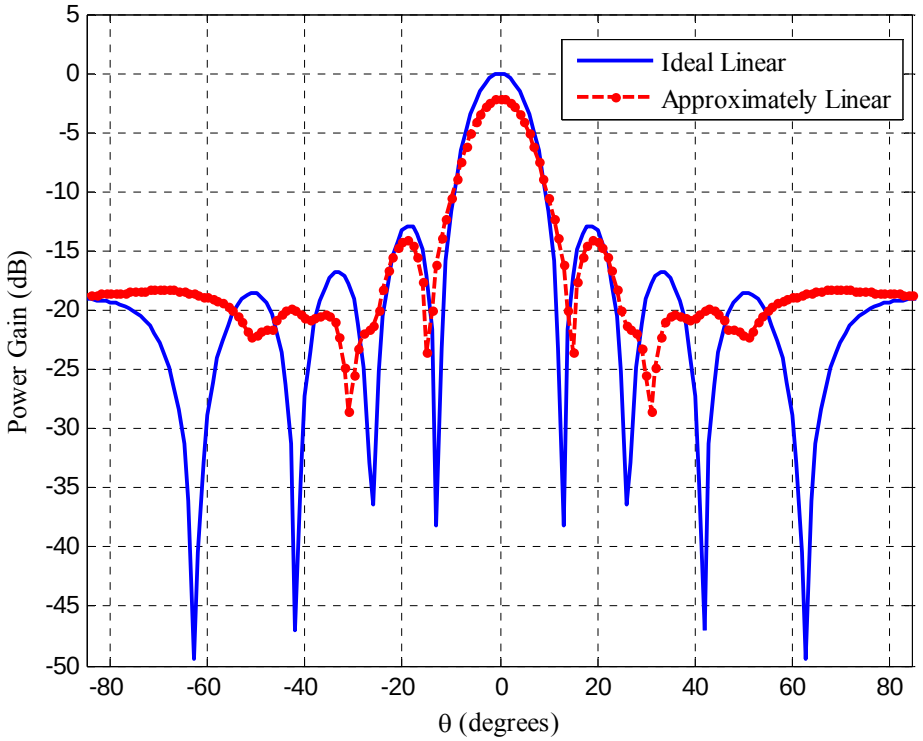

Figure 43. Average Beampattern for Uniform and Approximately Linear 9 Elements Array at 900 MHz for 600 Nodes for 1000 Monte Carlo Runs

D. PLANAR ANTENNA ARRAY RESULTS AND ANALYSIS

Planar array formation of 3×3 and 4×4 nodes was performed using Monte Carlo simulations of 100 runs for each case. The area of the sensor network was 10×10 meters^2 and the communication radius of each node was 4 meters. We used an operational frequency of 300 MHz, which results in an inter-node spacing of 0.5 meters. The simulation input data are shown in Table 8.

Simulations Input Data		
λ/2 (m)	0.5	0.5
Number of Runs (Monte Carlo)	100	100
Sensor Field Dimensions (m^2)	10×10	10×10
Clusterhead Radius (m)	4	4
Planar Array Formation (# of nodes)	3×3	4×4

Table 3. Simulation Parameters used for Planar Antenna Array Formation

1. Planar Array of 3×3 Nodes at 300 MHz

The first planar array formation simulation was performed for a 3×3 grid of 9 nodes for a variety of densities in an area of 10×10 meters^2. The array formation method used is the line fitting described in the previous chapter. The results for all densities are plotted in Figure 44. We can see that the total distance error is slightly below 0.1 meters for the higher densities.

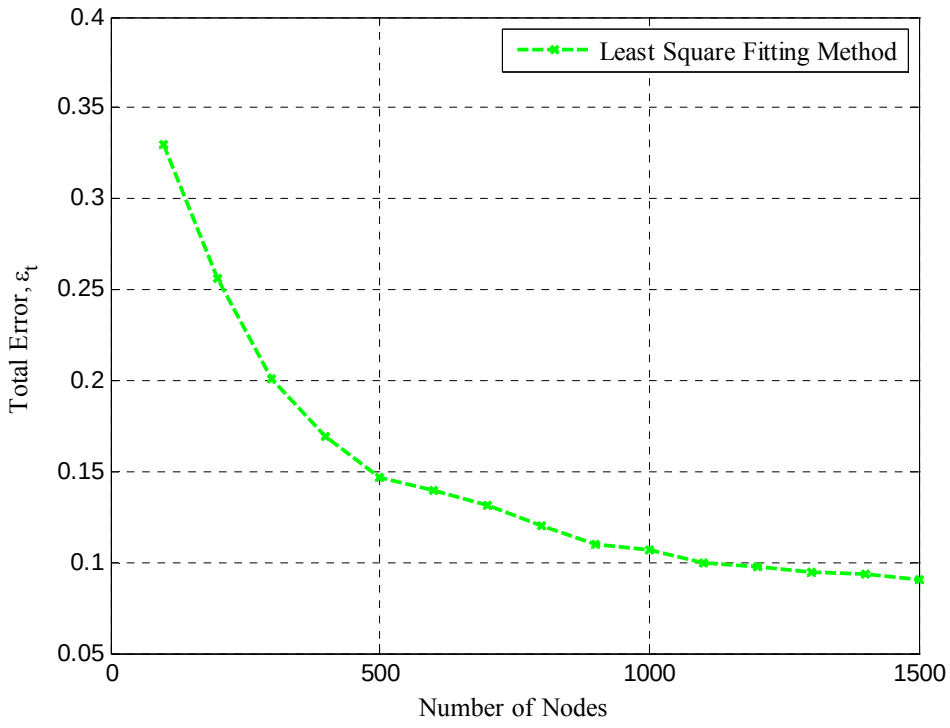

Figure 44. Average Total Error ε_t for 300 MHz and 3×3 Nodes Planar Antenna Array for 100 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

The average beampattern is plotted to show the effectiveness of the line fitting method in the construction of a planar array [17]. The average beampattern for node densities of 300 and 800 are presented in Figures 45 and 46, respectively. Comparing Figures 45 and 46, we observe an improvement in the side lobe level for the case of 800 nodes.

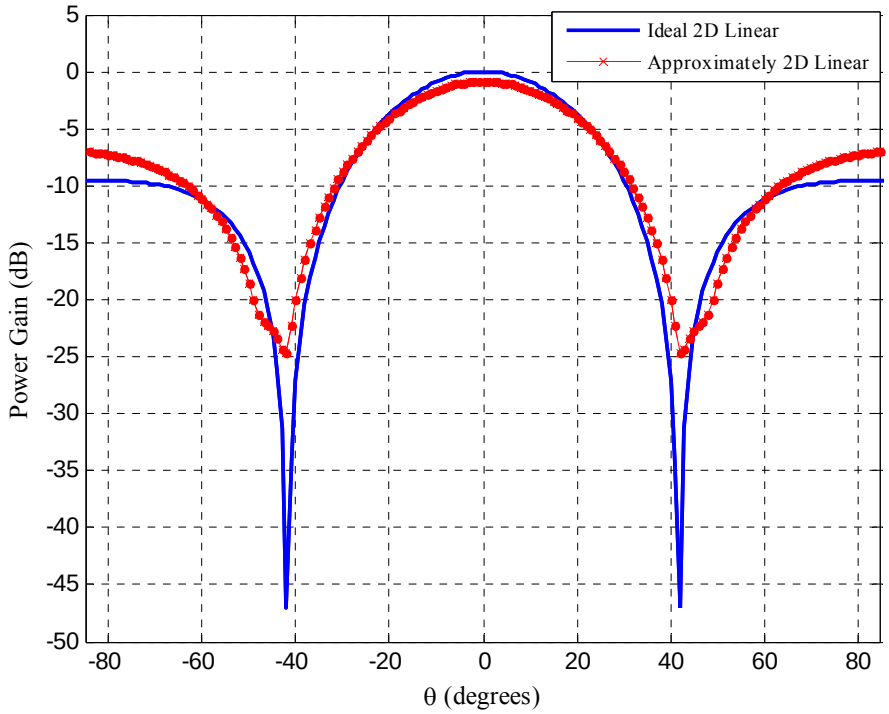

Figure 45. Average Beampattern for 3×3 Uniformly Excited Planar Array at 300 MHz for 300 Nodes for 100 Monte Carlo Runs

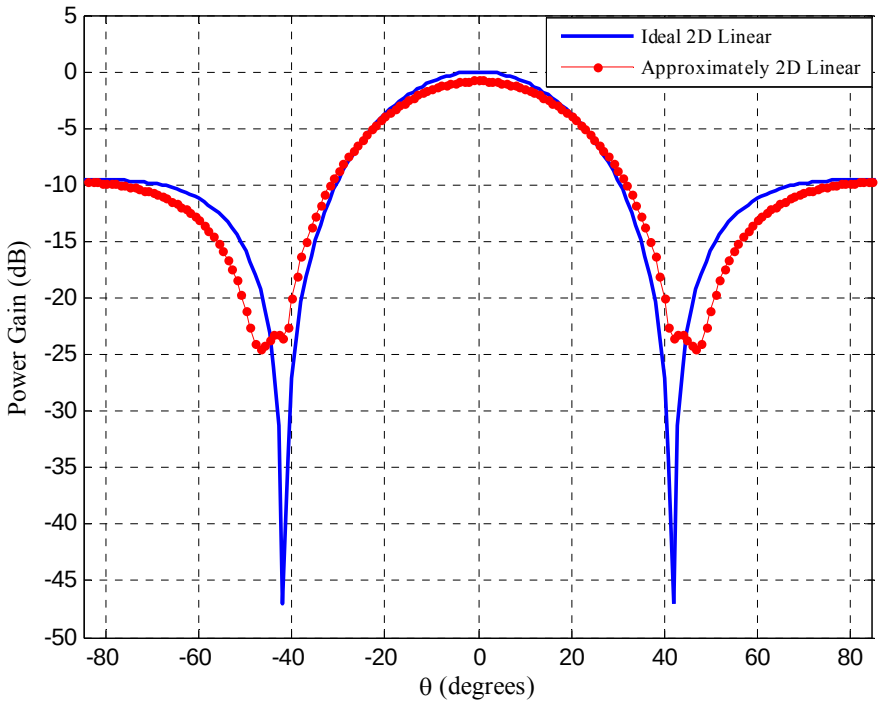

Figure 46. Average Beampattern for 3×3 Uniformly Excited Planar Array at 300 MHz for 800 Nodes for 100 Monte Carlo Runs

2. Planar Array of 4×4 Nodes at 300 MHz

The second planar array formation simulation performed was for a 4×4 grid of 16 nodes for a variety of densities in an area of 10×10 meters^2. The array formation method used was again line fitting. The total distance error is plotted in Figure 47.

Comparing Figures 44 and 47, the 3×3 and 4×4 planar arrays at the same frequency of 300 MHz, we observe slightly increased total error for the 4×4 array at all densities. The 3×3 planar array provides error below 0.1 m for high densities above 1100 nodes (see Figure 44). The 4×4 case in Figure 47 limits the error to just above 0.1 m for the highest density of 1500 nodes used in the simulations.

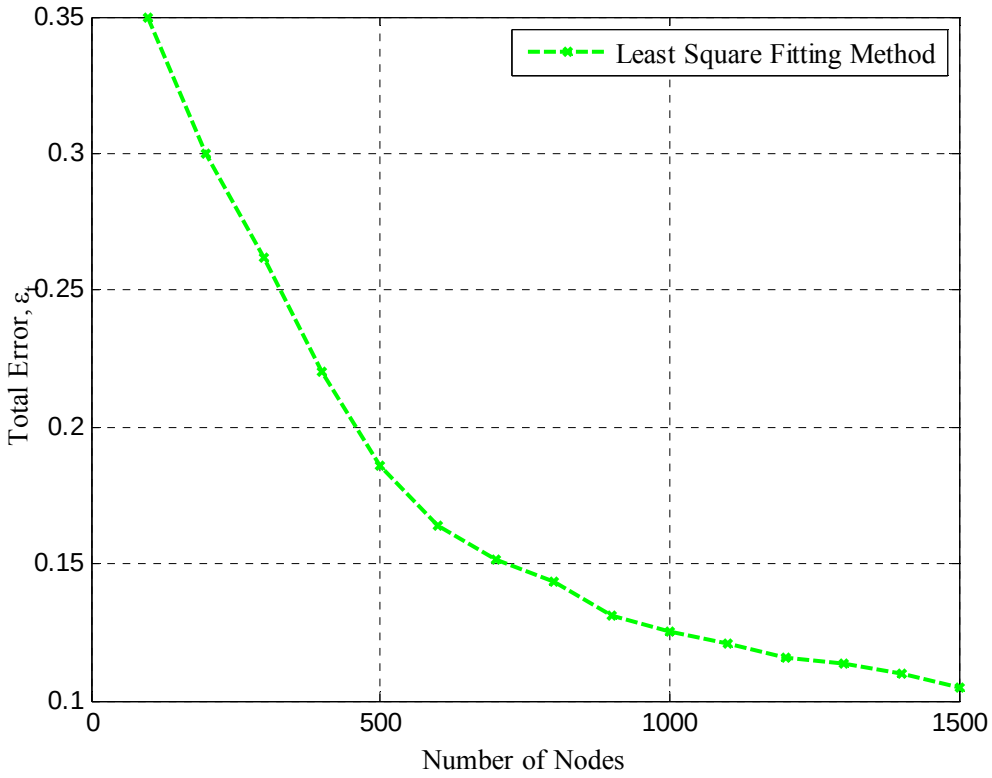

Figure 47. Average Total Error ε_t for 300 MHz and 4×4 Nodes Planar Antenna Array for 100 Monte Carlo Runs as a Function of the Number of Nodes in the Sensor Field

Average beampatterns for densities of 300 and 800 nodes are shown in Figures 48 and 49, respectively. In Figure 49, we observe a slight improvement in the main lobe as well as the side lobes for the 800-node case over the 300-node case.

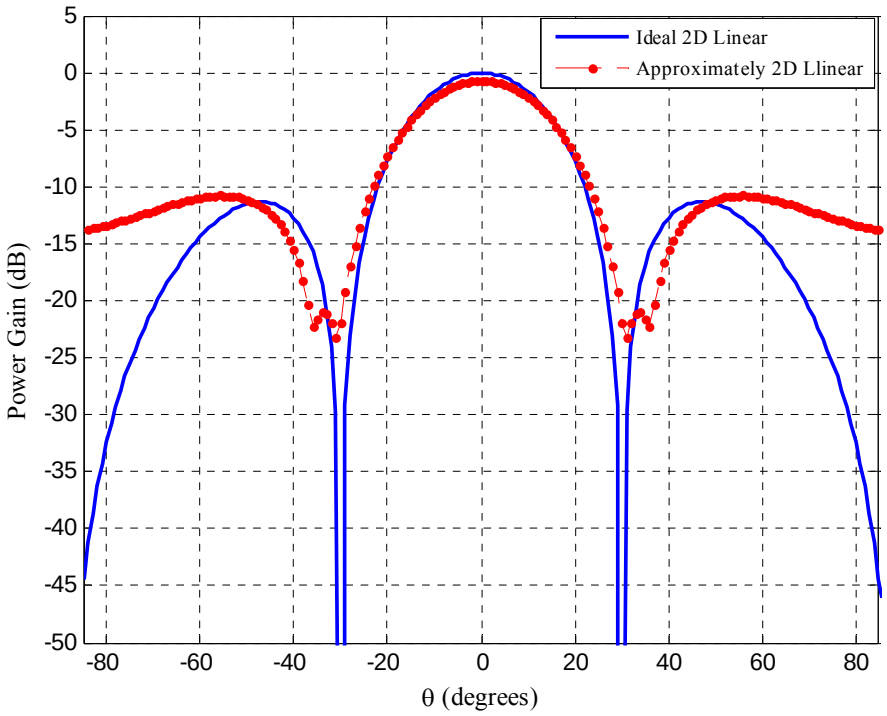

Figure 48. Average Beampattern for 4×4 Uniformly Excited Planar Array at 300 MHz for 300 Nodes for 100 Monte Carlo Runs

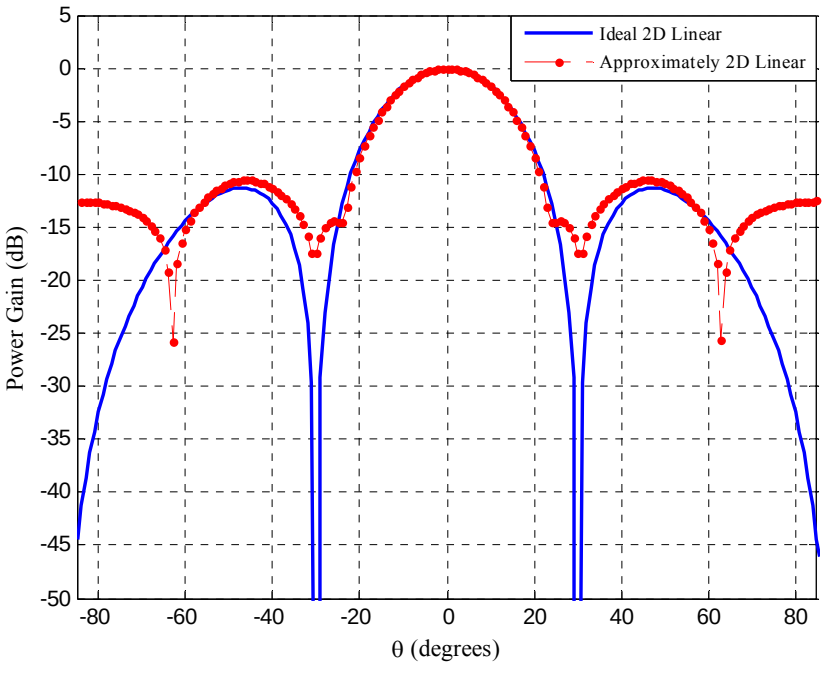

Figure 49. Average Beampattern for 4×4 Uniformly Excited Planar Array at 300 MHz for 800 Nodes for 100 Monte Carlo Runs

E. SUMMARY

In this chapter, we first presented the performance evaluation metrics for comparing the array formation methods. Three metrics used to evaluate the performance of the proposed linear array techniques were the perpendicular distance, the inter-node spacing and the total distance error. In the planar array formation, a total distance metric was used.

The simulation results were presented and demonstrated the effect of the sensor node density, frequency of operation, and number of elements in the construction of the array. The average error metrics in the construction of both the linear and planar arrays decreased as the density of nodes increased for all simulations. Also, the line fitting method provided improved performance for all cases compared with the iterative method. Additionally, for the same frequency, and as more nodes were added to the array, the total error decreased slightly with density for the line fitting method. Total error increased significantly for the iterative method for both operating frequencies as more nodes are added. Finally, for the same number of nodes in the array, the error metrics of the iterative method decreases with density while the line fitting method shows very little difference.

For the planar array formation, the line fitting method is used in all cases and the total error decreased when the node density in the area was increased. The 4×4 planar array provided slightly increased total error over the 3×3 planar array, but the additional nodes resulted in improved beampattern.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

Wireless sensor networking (WSN) is a relatively new field of research with many applications, both military and commercial. In the military, WSNs could be used in hostile environments to minimize the need for human presence. A wireless sensor network consists of a large number of small nodes that are deployed in an area of interest for collecting information. A subgroup of the deployed nodes will then coordinate their transmissions through beamforming to achieve improved gain. The information, collected by the WSN, is relayed to an unmanned aerial vehicle (UAV), which cooperates with the transmission beam of the network.

This work investigated the formation of arrays of nodes in a randomly deployed sensor field with the main objective of evaluating different approaches for linear and planar array formation. By selecting appropriate nodes in the sensor field, the goal of each approach was to form the best possible arrays in order to achieve increased beamforming gain, thus communication distances. A model was developed and implemented in MATLAB to simulate the linear and planar array formation. Results were provided to compare formation methods for both linear and planar arrays.

A. SIGNIFICANT RESULTS

Results showed that the proposed line fitting approach to array formation demonstrated significant performance improvement over the existing iterative approach to construct a linear array. Also, the average position errors decreased as the node density increased in the construction of a linear and/or planar array.

Higher operating frequencies made no significant difference in linearity of the array in the line fitting method but affected the error metrics of the iterative method. For the same operating frequency, the beampattern of the line fitting method improved as more elements are used in the array construction.

The line fitting approach was expanded to planar array formation. The simulations performed for 3×3 and 4×4 node arrays indicated that the average total error metric increased slightly in the 4×4 case, but the beampattern showed an improvement as more nodes were added to the array.

B. FUTURE WORK

It was assumed in this work that the nodes in the sensor field have only knowledge of their inter-node distances. The construction of a local coordination system and its effects on the construction of a linear and planar antenna array may be investigated.

The error ε measured in this thesis is based on geometrical calculations. A future investigation at pattern degradation in terms of electrical error $\frac{2\pi\varepsilon}{\lambda}$ may be examined. This would allow comparison of results at different frequencies using different number of elements and different sensor node densities.

The use of complex weights may be also investigated as a way to compensate for pattern degradation in uniformly excited arrays and to lower sidelobe level by tapering the excitation.

The sensor field was assumed to be a flat surface in this thesis. The effect of ground elevation on nodes' position errors was not examined in this study. The implementation and deployment of the methods proposed in this thesis extended to uneven surfaces is a natural next step in this research effort. Existing theory on the effect of random surface errors on antenna gain and sidelobe level [18] should be investigated to determine if it can be applied to sensor networks on an uneven surface.

APPENDIX

This Appendix includes MATLAB code used in this thesis work. The various functions used are briefly described. The main MATLAB program is *sensor_array.m,* which handles all the sub-programs described below.

Plot_nodes.m

This program simulates an area field (with given dimensions) in which the desired number of nodes are uniformly distributed.

Add_connectivity.m

This program selects the pairs of nodes that communicate among them, according a given communication radius for the nodes.

Find_clusterhead.m

This program counts the number of nodes that each node is connected with and one with the highest connections (node neighbors) is selected as the CH.

Three_element_center.m and *Three_element_end*

These two programs select the first three nodes, including the CH, subject to the criteria of linearity and inter-spacing distance. The first is when the CH in the center of three nodes and the second is when the CH is on one of the ends.

Add_array_node.m

This program selects the next nodes that satisfy the criteria of linearity and inter-node spacing. The selection is done for one node at the time.

Linear_line_fitting_comparison_iterative.m

Initially, this program selects an iterative solution to form the array and then controls the lines.m function. Next, it implements the line fitting approach to the nodes within the communication radius of the CH (cluster) and selects the nodes that are closest to the ideal positions. After the desired number of rotations, according to the resulting linearity error, the best line fitting solution is selected.

Line_fitting.m

This function evaluates and compares the selected linear arrays (as outlined in Chapter III) for both the line fitting method and the iterative approach.

Planar_array_iterative.m

This program is an extension of the Linear_least_square.m for the planar array. The best planar array is selected based on the total error metrics of multiple solutions. This function also calculates the Euclidean distance error for the planar array.

Planar_lines.m

This function implements the line fitting method and the rotations in the sensor field and provides the results to *Planar_array_iterative.m* for evaluation.

Beamf_comparison.m

This function computes the average beampatterns for the line fitting approach in comparison with the ideal arrays. [17]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Linear_line_fitting_comparison_iterative.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long
clear all
clc
close all

H=1;
for iii=1:H
n=200;      %number of nodes in the field
r=2.8;      %communication distance radius of each node
X=10;       %X and Y field dimensions
Y=10;
f=300;      %communication frequency among nodes
q=9;        %number of nodes in the array

[x_vals,y_vals] = plot_nodes(n,X,Y);

[connected, distance] = add_connectivity(n, r, x_vals, y_vals);

clusterhead = findclusterhead(connected, x_vals, y_vals, r, n);

%Start of iterative solution
[linear_array_3,d_3]  =  three_element_CH_center_v1(clusterhead,  x_vals,
y_vals,...
    connected, distance,f,q);

% Find a good linear array with clusterhead at end for q=3
[linear_array_4,d_4]   =   three_element_CH_end_v1(clusterhead,   x_vals,
y_vals,...
    connected, distance,f,q);

%Find the best 3 nodes linear array whith the constrain of lamda /2 and
annotate
%the array on the plot
if   d_3==inf & d_4 == inf
    h = errordlg(...
        'The algorithm can not determine a n-element array with the
current CH',...
        'Sensor Network')
elseif d_3 == inf
    antenna_array2 = linear_array_3;
elseif d_4==inf
    antenna_array2 = linear_array_4;
elseif d_3>d_4
    antenna_array2 = linear_array_4;
else
    antenna_array2 = linear_array_3;
end

%plot the first three nodes of the iterative solution
if (   min(d_3, d_4) ~= inf)
    plot( [ x_vals(antenna_array2(1)) x_vals(antenna_array2(2) )] , ...
```

```
        [ y_vals(antenna_array2(1))   y_vals(antenna_array2(2)) ],'rx-',
'LineWidth',2)
    plot( [ x_vals(antenna_array2(2)) x_vals(antenna_array2(3) )] , ...
        [ y_vals(antenna_array2(2))   y_vals(antenna_array2(3)) ],'rx-',
'LineWidth',2)
    hold on
X_values_anten2=[x_vals(antenna_array2(1)),   x_vals(antenna_array2(2)),
x_vals(antenna_array2(3))];
Y_values_anten2=[y_vals(antenna_array2(1)),   y_vals(antenna_array2(2)),
y_vals(antenna_array2(3))];
format short
 end

%add one node at a time up to desired
array_size=9;
for i = 4:1:array_size
   [antenna_array2,  flag,  d_tempf,  X_values_anten2,  Y_values_anten2] =
add_array_node(antenna_array2, x_vals, y_vals,...
    connected, distance,f, X_values_anten2, Y_values_anten2);
format short

X_values_anten2;
Y_values_anten2;

end
hold on
freq=f;
intd= (((3*10^8)/(freq*10^6)))/2 ;
figure(2)
plot( x_vals(clusterhead), y_vals(clusterhead), 'ro', 'LineWidth',1,...
    'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8)
hold on

clusterhead_neighbors = find( connected( clusterhead,:) == 1);

index=size(clusterhead_neighbors,2);
indexx=(1+size(clusterhead_neighbors,2));
daspect('manual')
daspect([1 1 1])
axis([0 X 0 Y])
h = gcf;
rect = [50, 65, 650, 620];
set(h, 'Position', rect);

plot(    x_vals(clusterhead_neighbors),    y_vals(clusterhead_neighbors),
'ko')
 for i = 1:1:index
     text(  x_vals(i)+(X*10),y_vals(i)+(Y*1000),int2str(i),  'FontSize',
8);
 end

hold on
%Start of least squares line fitting solution
X_col=[x_vals(clusterhead_neighbors) x_vals(clusterhead)]';
Y_col=[y_vals(clusterhead_neighbors) y_vals(clusterhead)]';
```

```
XX=[ones(indexx,1) X_col];
XX2=[X_col];
YY=[Y_col];
c=XX\YY;
arith=abs(c(2)*XX2-YY+c(1));
paran=sqrt(c(2)^2+1);
d=arith/paran;
f=mean(d);

t=0:0.1:10;
z=c(2)*t+c(1);

k=[-(1/c(2)); y_vals(clusterhead)-(-1/c(2))*(x_vals(clusterhead))] ;
r=k(1)*t+k(2);
arithh=abs(k(1)*XX2-YY+k(2));
parann=sqrt(k(1)^2+1);
dd=arithh/parann;
ff=mean(dd);
plot(t,z,'r-',t,r,'b-')

hold on
%Plot least square fitting and perpendicular lines left an right
zz1=c(2)*t+c(1)+intd*(sqrt(1+(c(2))^2));
zz2=c(2)*t+c(1)-intd*(sqrt(1+(c(2))^2));
rr1=k(1)*t+k(2)+intd*(sqrt(1+(k(1))^2));
rr2=k(1)*t+k(2)-intd*(sqrt(1+(k(1))^2));
hold on
plot(t,zz1,'r--',t,zz2,'r--');
hold on
plot(t,rr1,'b--',t,rr2,'b--');

connected = zeros(index);
distance = inf * ones(index);

for i = 1:1:(indexx)
    for j = (i+1):1:indexx
            connected(i,j) = 1;
            distance(i,j)   =   sqrt(    (x_vals(i)-x_vals(j))^2      +
(y_vals(i)-y_vals(j))^2 );
            d_epalithesh(i)=arith(i)/paran;
    end

end

%FIND EDGES ON THE PARALLILOGRAM

%Ast point
xa=0;
ya=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1); k(2)];
XYA=[xa ya];
XYA=inv(A)*B
```

```
%Bth point
xd=0;
yd=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+intd*(sqrt(1+(c(2))^2)); k(2)];
XYB=[xd yd];
XYB=inv(A)*B

%2nd point
xb=0;
yb=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-intd*(sqrt(1+(c(2))^2)); k(2)];
XYC=[xb yb];
XYC=inv(A)*B

%Bth point
xd=0;
yd=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+2*intd*(sqrt(1+(c(2))^2)); k(2)];
XYD=[xd yd];
XYD=inv(A)*B

%2nd point
xb=0;
yb=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-2*intd*(sqrt(1+(c(2))^2)); k(2)];
XYE=[xb yb];
XYE=inv(A)*B

%Bth point
xd=0;
yd=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+3*intd*(sqrt(1+(c(2))^2)); k(2)];
XYF=[xd yd];
XYF=inv(A)*B

%2nd point
xb=0;
yb=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-3*intd*(sqrt(1+(c(2))^2)); k(2)];
XYG=[xb yb];
XYG=inv(A)*B


%Bth point
xd=0;
yd=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+4*intd*(sqrt(1+(c(2))^2)); k(2)];
XYH=[xd yd];
```

```
XYH=inv(A)*B

%2nd point
xb=0;
yb=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-4*intd*(sqrt(1+(c(2))^2)); k(2)];
XYI=[xb yb];
XYI=inv(A)*B

plot(XYA(2),XYA(1),   'bx',   XYB(2),   XYB(1),   'bx',XYC(2),   XYC(1),
'bx',XYD(2), XYD(1), 'bx',...
    XYE(2),   XYE(1),   'bx',XYF(2),   XYF(1),   'bx',XYG(2),   XYG(1),
'bx',XYH(2), XYH(1), 'bx',XYI(2), XYI(1), 'bx')

vv1=Inf
for i=1:index
    db=sqrt((XYB(2)-x_vals(clusterhead_neighbors(i)))^2+(XYB(1)-
y_vals(clusterhead_neighbors(i)))^2)

    if db<vv1
        vv1=db
        end1=i
    end
end
vv2=Inf
for i=1:index
    if i~=end1
    dc=sqrt((XYC(2)-x_vals(clusterhead_neighbors(i)))^2+(XYC(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dc<vv2
        vv2=dc
        end2=i
    end
    end
end

vv3=Inf
for i=1:index
    if i~=end1 & i~=end2
    dd=sqrt((XYD(2)-x_vals(clusterhead_neighbors(i)))^2+(XYD(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dd<vv3
        vv3=dd
        end3=i
    end
    end
end
vv4=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3
    de=sqrt((XYE(2)-x_vals(clusterhead_neighbors(i)))^2+(XYE(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if de<vv4
        vv4=de
        end4=i
```

```
    end
    end
end

vv5=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4
    df=sqrt((XYF(2)-x_vals(clusterhead_neighbors(i)))^2+(XYF(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if df<vv5
        vv5=df
        end5=i
    end
    end
end

vv6=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5
    dg=sqrt((XYG(2)-x_vals(clusterhead_neighbors(i)))^2+(XYG(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dg<vv6
        vv6=dg
        end6=i
    end
    end
end

vv7=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6
    dh=sqrt((XYH(2)-x_vals(clusterhead_neighbors(i)))^2+(XYH(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dh<vv7
        vv7=dh
        end7=i
    end
    end
end

vv8=Inf
for i=1:index
    if  i~=end1  &  i~=end2  &  i~=end3  &  i~=end4  &  i~=end5  &  i~=end6
i~=end7
    di=sqrt((XYI(2)-x_vals(clusterhead_neighbors(i)))^2+(XYI(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if di<vv8
        vv8=di
        end8=i
    end
    end
end

d_total1=vv1+vv2+vv3+vv4+vv5+vv6+vv7+vv8
%plot first solution
plot( [ x_vals(clusterhead) x_vals(clusterhead_neighbors(end1)) ] , ...
```

```
      [ y_vals(clusterhead)   y_vals(clusterhead_neighbors(end1)) ],'go-
', 'LineWidth',2)
plot( [ x_vals(clusterhead) x_vals(clusterhead_neighbors(end2)) ] , ...
      [ y_vals(clusterhead)   y_vals(clusterhead_neighbors(end2)) ],'go-
', 'LineWidth',2)
plot(                [                x_vals(clusterhead_neighbors(end1))
x_vals(clusterhead_neighbors(end3) )] , ...
      [                                y_vals(clusterhead_neighbors(end1))
y_vals(clusterhead_neighbors(end3)) ],'go-', 'LineWidth',2)
  plot(              [                x_vals(clusterhead_neighbors(end2))
x_vals(clusterhead_neighbors(end4) )] , ...
      [                                y_vals(clusterhead_neighbors(end2))
y_vals(clusterhead_neighbors(end4)) ],'go-', 'LineWidth',2)
  plot(              [                x_vals(clusterhead_neighbors(end3))
x_vals(clusterhead_neighbors(end5) )] , ...
      [                                y_vals(clusterhead_neighbors(end3))
y_vals(clusterhead_neighbors(end5)) ],'go-', 'LineWidth',2)
  plot(              [                x_vals(clusterhead_neighbors(end4))
x_vals(clusterhead_neighbors(end6) )] , ...
      [                                y_vals(clusterhead_neighbors(end4))
y_vals(clusterhead_neighbors(end6)) ],'go-', 'LineWidth',2)
  plot(              [                x_vals(clusterhead_neighbors(end5))
x_vals(clusterhead_neighbors(end7) )] , ...
      [                                y_vals(clusterhead_neighbors(end5))
y_vals(clusterhead_neighbors(end7)) ],'go-', 'LineWidth',2)
  plot(              [                x_vals(clusterhead_neighbors(end6))
x_vals(clusterhead_neighbors(end8) )] , ...
      [                                y_vals(clusterhead_neighbors(end6))
y_vals(clusterhead_neighbors(end8)) ],'go-', 'LineWidth',2)

      hold on

%Second solution on rotated lines
      %Ast point
xa=0;
ya=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1); k(2)];
XYAA=[xa ya];
XYAA=inv(A)*B

 %B point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)+intd*(sqrt(1+(k(1))^2)); c(1)];
XYBB=[xd yd];
XYBB=inv(A)*B

 %C point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)-intd*(sqrt(1+(k(1))^2)); c(1)];
XYCC=[xd yd];
XYCC=inv(A)*B
```

```
 %D point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)+2*intd*(sqrt(1+(k(1))^2)); c(1)];
XYDD=[xd yd];
XYDD=inv(A)*B

 %E point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)-2*intd*(sqrt(1+(k(1))^2)); c(1)];
XYEE=[xd yd];
XYEE=inv(A)*B

%F point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)+3*intd*(sqrt(1+(k(1))^2)); c(1)];
XYFF=[xd yd];
XYFF=inv(A)*B

%G point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)-3*intd*(sqrt(1+(k(1))^2)); c(1)];
XYGG=[xd yd];
XYGG=inv(A)*B

%H point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)+4*intd*(sqrt(1+(k(1))^2)); c(1)];
XYHH=[xd yd];
XYHH=inv(A)*B

% I point
xd=0;
yd=0;
A=[1 -k(1); 1 -c(2)];
B=[k(2)-4*intd*(sqrt(1+(k(1))^2)); c(1)];
XYII=[xd yd];
XYII=inv(A)*B

hold on
plot(XYAA(2),XYAA(1),  'rx',  XYBB(2),  XYBB(1),  'rx',XYCC(2),  XYCC(1),
'rx',XYDD(2), XYDD(1), 'rx',...
    XYEE(2),  XYEE(1),  'rx',XYFF(2),  XYFF(1),  'rx',XYGG(2),  XYGG(1),
'rx',XYHH(2), XYHH(1), 'rx',XYII(2), XYII(1), 'rx')
```

```
%end1=0
vvv1=Inf
for i=1:index
    dbb=sqrt((XYBB(2)-x_vals(clusterhead_neighbors(i)))^2+(XYBB(1)-
y_vals(clusterhead_neighbors(i)))^2)

    if dbb<vvv1
        vvv1=dbb
        end11=i
    end
end
vvv2=Inf
for i=1:index
    if i~=end11
    dcc=sqrt((XYCC(2)-x_vals(clusterhead_neighbors(i)))^2+(XYCC(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dcc<vvv2
        vvv2=dcc
        end22=i
    end
    end
end

vvv3=Inf
for i=1:index
    if i~=end11 & i~=end22
    ddd=sqrt((XYDD(2)-x_vals(clusterhead_neighbors(i)))^2+(XYDD(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if ddd<vvv3
        vvv3=ddd
        end33=i
    end
    end
end
vvv4=Inf
for i=1:index
    if i~=end11 & i~=end22 & i~=end33
    dee=sqrt((XYEE(2)-x_vals(clusterhead_neighbors(i)))^2+(XYEE(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dee<vvv4
        vvv4=dee
        end44=i
    end
    end
end

vvv5=Inf
for i=1:index
    if i~=end11 & i~=end22 & i~=end33 & i~=end44
    dff=sqrt((XYFF(2)-x_vals(clusterhead_neighbors(i)))^2+(XYFF(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dff<vvv5
        vvv5=dff
        end55=i
    end
    end
```

```
end

vvv6=Inf
for i=1:index
    if i~=end11 & i~=end22 & i~=end33 & i~=end44 & i~=end55
    dgg=sqrt((XYGG(2)-x_vals(clusterhead_neighbors(i)))^2+(XYGG(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dgg<vvv6
        vvv6=dgg
        end66=i
    end
    end
end

vvv7=Inf
for i=1:index
    if i~=end11 & i~=end22 & i~=end33 & i~=end44 & i~=end55 & i~=end66
    dhh=sqrt((XYHH(2)-x_vals(clusterhead_neighbors(i)))^2+(XYHH(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dhh<vvv7
        vvv7=dhh
        end77=i
    end
    end
end

vvv8=Inf
for i=1:index
    if i~=end11 & i~=end22 & i~=end33 & i~=end44 & i~=end55 & i~=end66
i~=end77
    dii=sqrt((XYII(2)-x_vals(clusterhead_neighbors(i)))^2+(XYII(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dii<vvv8
        vvv8=dii
        end88=i
    end
    end
end

d_total1b=vvv1+vvv2+vvv3+vvv4+vvv5+vvv6+vvv7+vvv8
%plot second solution
plot(  [  x_vals(clusterhead)  x_vals(clusterhead_neighbors(end11))  ]  ,
...
      [   y_vals(clusterhead)      y_vals(clusterhead_neighbors(end11))
],'go-', 'LineWidth',2)
plot(  [  x_vals(clusterhead)  x_vals(clusterhead_neighbors(end22))  ]  ,
...
      [   y_vals(clusterhead)      y_vals(clusterhead_neighbors(end22))
],'go-', 'LineWidth',2)
plot(               [               x_vals(clusterhead_neighbors(end11))
x_vals(clusterhead_neighbors(end33) )] , ...
      [                             y_vals(clusterhead_neighbors(end11))
y_vals(clusterhead_neighbors(end33)) ],'go-', 'LineWidth',2)
  plot(             [             x_vals(clusterhead_neighbors(end22))
x_vals(clusterhead_neighbors(end44) )] , ...
```

```
      [                              y_vals(clusterhead_neighbors(end22))
y_vals(clusterhead_neighbors(end44)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end33))
x_vals(clusterhead_neighbors(end55) )] , ...
      [                              y_vals(clusterhead_neighbors(end33))
y_vals(clusterhead_neighbors(end55)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end44))
x_vals(clusterhead_neighbors(end66) )] , ...
      [                              y_vals(clusterhead_neighbors(end44))
y_vals(clusterhead_neighbors(end66)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end55))
x_vals(clusterhead_neighbors(end77) )] , ...
      [                              y_vals(clusterhead_neighbors(end55))
y_vals(clusterhead_neighbors(end77)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end66))
x_vals(clusterhead_neighbors(end88) )] , ...
      [                              y_vals(clusterhead_neighbors(end66))
y_vals(clusterhead_neighbors(end88)) ],'go-', 'LineWidth',2)

      hold off

hold off
      hold off

     DDa_total=min(d_total1, d_total1b)

     if DDa_total==d_total1
         X_vals_linear_array=[x_vals(clusterhead_neighbors(end7)),
x_vals(clusterhead_neighbors(end5)),
x_vals(clusterhead_neighbors(end3)),
x_vals(clusterhead_neighbors(end1)),               x_vals(clusterhead),
x_vals(clusterhead_neighbors(end2)),
x_vals(clusterhead_neighbors(end4)),
x_vals(clusterhead_neighbors(end6)),x_vals(clusterhead_neighbors(end8))
]
         Y_vals_linear_array=[y_vals(clusterhead_neighbors(end7)),
y_vals(clusterhead_neighbors(end5)),
y_vals(clusterhead_neighbors(end3)),
y_vals(clusterhead_neighbors(end1)),               y_vals(clusterhead),
y_vals(clusterhead_neighbors(end2)),
y_vals(clusterhead_neighbors(end4)),
y_vals(clusterhead_neighbors(end6)),y_vals(clusterhead_neighbors(end8))
]


     elseif DDa_total==d_total1b
         X_vals_linear_array=[x_vals(clusterhead_neighbors(end77)),
x_vals(clusterhead_neighbors(end55)),
x_vals(clusterhead_neighbors(end33)),
x_vals(clusterhead_neighbors(end11)),               x_vals(clusterhead),
x_vals(clusterhead_neighbors(end22)),
x_vals(clusterhead_neighbors(end44)),
x_vals(clusterhead_neighbors(end66)),x_vals(clusterhead_neighbors(end88
))]
        Y_vals_linear_array=[y_vals(clusterhead_neighbors(end77)),
y_vals(clusterhead_neighbors(end55)),
```

```
y_vals(clusterhead_neighbors(end33)),
y_vals(clusterhead_neighbors(end11)),              y_vals(clusterhead),
y_vals(clusterhead_neighbors(end22)),
y_vals(clusterhead_neighbors(end44)),
y_vals(clusterhead_neighbors(end66)),y_vals(clusterhead_neighbors(end88
))]

   end
 ANTENNA2(iii,:)=antenna_array2
  X_VALUES_ANTENNA2(iii,:)=X_values_anten2
  Y_VALUES_ANTENNA2(iii,:)=Y_values_anten2

X_vals_Linear_array_trend(iii, :)=X_vals_linear_array
Y_vals_Linear_array_trend(iii, :)=Y_vals_linear_array

hold off
 end
%best  solutions  of  iterative  and  least  squares  line  fitting  are
evaluated at
%linear_fitting function
  [     mean_dd,     mean_dd2,     Int_mean,     Int2_mean,     Tot_mean,
Tot2_mean]=linnear_fitting9(...
    X_vals_Linear_array_trend,               Y_vals_Linear_array_trend,
X_VALUES_ANTENNA2, Y_VALUES_ANTENNA2);

clc
X_VALUES_ANTENNA2(iii,:)=X_values_anten2
Y_VALUES_ANTENNA2(iii,:)=Y_values_anten2

X_vals_Linear_array_trend(iii, :)=X_vals_linear_array
Y_vals_Linear_array_trend(iii, :)=Y_vals_linear_array
 mean_dd
 mean_dd2
 Int_mean
 Int2_mean
 Tot_mean
 Tot2_mean
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Linnear_fitting.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function   [   mean_dd,   mean_dd2,   Int_mean,   Int2_mean,   Tot_mean,
Tot2_mean]=linnear_fitting(...
    x_vals_Linear_array_trend,                y_vals_Linear_array_trend,
X_VALUES_ANTENNA2, Y_VALUES_ANTENNA2)
GG=1000
H=GG;   %Number of iteration to be tested for errors
K=GG;
G=GG;
T=GG;
 for i=1:H

 f=300;
dddd = (((3*10^8)/(f*10^6)))/2 ;

%x,y are the one step calculation of nodes array
%x2,y2 are the multible steps calculation of node array

x=x_vals_Linear_array_trend;
y=y_vals_Linear_array_trend;
x2=X_VALUES_ANTENNA2;
y2=Y_VALUES_ANTENNA2;

% Get one line each time and find the error
x=x(i,:);
y=y(i,:);
x2=x2(i,:);
y2=y2(i,:);

x=x';
y=y';
x2=x2';
y2=y2';

%Perpedicular distance error
X=[ones(9,1) x];    %9 for 9-node array or 7 for 7-node array
Y=y;
c=X\Y;
t=0:0.1:10;
z=c(2)*t+c(1);

for j=1:9          %9 for 9-node array or 7 for 7-node array

arith(j)=abs(c(2)*x(j)-y(j)+c(1));
paran(j)=sqrt((c(2))^2+1);
d(j)=arith(j)/paran(j);
t=0:0.1:10;
z=c(2)*t+c(1);
k=[-(1/c(2)); y(j)+(1/c(2))*x(j)] ;
r=k(1)*t+k(2);

xx(j)=(k(2)-c(1))/(c(2)-k(1));
```

```
yy(j)=k(1)*(xx(j))+k(2);

 end

x=x';
y=y';
X2=[ones(9,1) x2];
Y2=y2;
c2=X2\Y2;
t=0:0.1:10;
z2=c2(2)*t+c2(1);

for j=1:9
arith2(j)=abs(c2(2)*x2(j)-y2(j)+c2(1));
paran2(j)=sqrt((c2(2))^2+1);
d2(j)=arith2(j)/paran2(j);
t=0:0.1:10;
z2=c2(2)*t+c2(1);
k2=[-(1/c2(2)); y2(j)+(1/c2(2))*x2(j)] ;
r2=k2(1)*t+k2(2);
xx2(j)=(k2(2)-c2(1))/(c2(2)-k2(1));
yy2(j)=k2(1)*(xx2(j))+k2(2);
 end

x2=x2';
y2=y2';
x=x';
y=y';
Xxx(i,:)=xx;
Yyy(i,:)=yy;
Xxx2(i,:)=xx2;
Yyy2(i,:)=yy2;

dd(i,:)=d;
dd2(i,:)=d2;
end

 dd;
 dd2;
 m_dd=mean(dd');
 m_dd2=mean(dd2');
 mean_dd=mean(m_dd)
 mean_dd2=mean(m_dd2)

 %Inter-node distance error
 for g=1:G
 xx=Xxx(g,:);
 yy=Yyy(g,:);
 xx2=Xxx2(g,:);
 yy2=Yyy2(g,:);

for jj=1:8
    prod(jj)=sqrt((xx(1)-xx(jj+1))^2+(yy(1)-yy(jj+1))^2);
    prod2(jj)=sqrt((xx2(1)-xx2(jj+1))^2+(yy2(1)-yy2(jj+1))^2);
end
```

```
 prodd(g,:)=prod;
 prodd2(g,:)=prod2;
 end

 prodd;
 prodd2;
 pro=mean(prodd);
 pro2=mean(prodd2);
 pro_mean=mean(pro);
 pro2_mean=mean(pro2);

  for k=1:K

     pr=prodd(k,:);
     pr2=prodd2(k,:);

     for jj=1:8
         interd(jj)=abs(((jj)*dddd)-(pr(jj)));
          interd2(jj)=abs(((jj)*dddd)-(pr2(jj)));
     end

     Interdd(k,:)=interd;
     Interdd2(k,:)=interd2;
 end
 Interdd;
 Interdd2;
 Int=mean(Interdd');
 Int2=mean(Interdd2');
 Int_mean=mean(Int);
 Int2_mean=mean(Int2);

%Total error

 for t=1:T

     pr=prodd(t,:);
     pr2=prodd2(t,:);
     Int=Interdd(t,:);
     Int2=Interdd2(t,:);
     dd_d=dd(t,:);
     dd2_d=dd2(t,:);

     for jj=1:8
         tot(jj)=sqrt((dd_d(jj))^2+(Int(jj))^2);
         tot2(jj)=sqrt((dd2_d(jj))^2+(Int2(jj))^2);
     end
     Total(t,:)=tot;
     Total2(t,:)=tot2;
 end
         Total;
         Total2;
         Tot=mean(Total');
         Tot2=mean(Total2');
         Tot_mean=mean(Tot);  Tot2_mean=mean(Tot2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Planar_array_iterative.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format short
clear all
clc
close all

H=2;
for iii=1:H

n=900;   %insert the data
r=1.5;
X=10;
Y=10;

q=7;
freq=300;
intd= (((3*10^8)/(freq*10^6)))/2 ;


%Plot nodes
x_vals = rand( 1 , n);
y_vals = rand( 1 , n);


x_vals = x_vals * X;
y_vals = y_vals * Y;

figure(1)
plot( x_vals, y_vals, 'ko')

daspect('manual')
daspect([1 1 1])
axis([0 X 0 Y])

h = gcf;
rect = [50, 65, 650, 620];
set(h, 'Position', rect);


for i = 1:1:n
    text( x_vals(i)+(X/55),y_vals(i)+(Y/55),int2str(i), 'FontSize', 8)
end
hold on


%Add connectivity
connected = zeros(n);
distance = inf * ones(n);


for i = 1:1:(n-1)
```

```
    for j = (i+1):1:n
        if  (      (  (x_vals(i)  -  x_vals(j))^2    +    (y_vals(i)  -
y_vals(j))^2 ) < r*r )
            connected(i,j) = 1;
            distance(i,j)    =    sqrt(    (x_vals(i)-x_vals(j))^2       +
(y_vals(i)-y_vals(j))^2 );
        end
    end
end

for i = 1:1:(n-1)
    for j = (i+1):1:n
        if (   connected(i,j) == 1)
            plot(   [ x_vals(i)   x_vals(j)]   ,    [ y_vals(i)   y_vals(j)
]   )
        end
    end
end

for i = 2:1:n
    for j = 1:1:(i-1)
        connected(i,j) = connected(j,i);
        distance(i,j)=distance(j,i);
    end
end

%Find clusterhead

row_count_dok = zeros(1,n);
TT=0;
for i=1:1:n
    for j=(i+1):1:n
    if (    ( (x_vals(i) - x_vals(j))^2  +  (y_vals(i) - y_vals(j))^2 )
< r*r )
        row_count_dok(i)= row_count_dok(i) + 1;

        end
    end
    end
H=max(row_count_dok)
    % max(row_count_dok)
for i = 1:n
    if row_count_dok(i) == H
        clusterhead = i;
    end
end


plot( x_vals(clusterhead), y_vals(clusterhead), 'ro', 'LineWidth',1,...
    'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8)

clusterhead_neighbors = find( connected( clusterhead,:) == 1);

index=size(clusterhead_neighbors,2);
indexx=(1+size(clusterhead_neighbors,2));
```

```
%New figure only with clusterhead and clusterhead_neighbors
hold off
figure(2)
plot( x_vals(clusterhead), y_vals(clusterhead), 'ro', 'LineWidth',1,...
    'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8)
hold on


daspect('manual')
daspect([1 1 1])
axis([0 X 0 Y])

h = gcf;
rect = [50, 65, 650, 620];
set(h, 'Position', rect);

plot(    x_vals(clusterhead_neighbors),    y_vals(clusterhead_neighbors),
'ko')
 for i = 1:1:index
     text(  x_vals(i)+(X*10),y_vals(i)+(Y*1000),int2str(i),  'FontSize',
8);
 end

hold on

%Plot least square fitting and perpendicular lines left an right
X_col=[x_vals(clusterhead_neighbors) x_vals(clusterhead)]';
Y_col=[y_vals(clusterhead_neighbors) y_vals(clusterhead)]';

XX=[ones(indexx,1) X_col];
XX2=[X_col];
YY=[Y_col];
c=XX\YY;

%function that finds the solution and evaluates the results
[planar] = lines (clusterhead, connected, x_vals, y_vals, c ,XX2, YY,
intd);
hold off
for i=1:6
figure(i+1)
plot( x_vals(clusterhead), y_vals(clusterhead), 'ro', 'LineWidth',1,...
    'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8)
hold on
daspect('manual')
daspect([1 1 1])
axis([0 X 0 Y])

h = gcf;
rect = [50, 65, 650, 620];
set(h, 'Position', rect);

plot(    x_vals(clusterhead_neighbors),    y_vals(clusterhead_neighbors),
'ko')
```

```
hold on
c(2)=c(2)-tan(pi/6)
c(1)=y_vals(clusterhead)-(c(2))*x_vals(clusterhead)

[d_total1] = Planar_lines (clusterhead, connected, x_vals, y_vals, c
,XX2, YY, intd);

d_total1
hold off
D_total1(i,:)=d_total1

end

DD_total1=min(D_total1)
DDD_total1(iii,:)=DD_total1
End
%results for the planar array
Final_total1=min(DDD_total1)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Planar_lines.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [planar] = Planar_lines (clusterhead, connected, x_vals, ...
    y_vals, c ,XX2, YY, intd)

clusterhead_neighbors = find( connected( clusterhead,:) == 1);
index=size(clusterhead_neighbors,2);
indexx=(1+size(clusterhead_neighbors,2));
arith=abs(c(2)*XX2-YY+c(1));
paran=sqrt(c(2)^2+1);
d=arith/paran;
f=mean(d);

t=0:0.1:10;
z=c(2)*t+c(1);
%start to construct the lines and points
k=[-(1/c(2)); y_vals(clusterhead)-(-1/c(2))*(x_vals(clusterhead))] ;
r=k(1)*t+k(2);
arithh=abs(k(1)*XX2-YY+k(2));
parann=sqrt(k(1)^2+1);
dd=arithh/parann;
ff=mean(dd);
plot(t,z,'r-',t,r,'b-')

hold on
zz1=c(2)*t+c(1)+intd*(sqrt(1+(c(2))^2));
zz2=c(2)*t+c(1)-intd*(sqrt(1+(c(2))^2));
zz3=c(2)*t+c(1)+2*intd*(sqrt(1+(c(2))^2));
zz4=c(2)*t+c(1)-2*intd*(sqrt(1+(c(2))^2));
rr1=k(1)*t+k(2)+intd*(sqrt(1+(k(1))^2));
rr2=k(1)*t+k(2)-intd*(sqrt(1+(k(1))^2));
rr3=k(1)*t+k(2)+2*intd*(sqrt(1+(k(1))^2));

hold on
plot(t,zz1,'r--',t,zz2,'r--');
hold on
plot(t,rr1,'b--',t,rr2,'b--');
hold on
plot(t,zz3,'r--');
hold on
plot(t,rr3,'b--');

connected = zeros(index);
distance = inf * ones(index);
 for i = 1:1:(indexx)
    for j = (i+1):1:indexx
            connected(i,j) = 1;
            distance(i,j) = sqrt( (x_vals(i)-x_vals(j))^2 +…
                            (y_vals(i)-y_vals(j))^2 );
            d_epalithesh(i)=arith(i)/paran;
            %kk(i)=arithh(i)/parann;
    end
  end
```

```
%FIND EDGES ON THE PARALLILOGRAM
%1st point in the center
xa=0;
ya=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1); k(2)];
XYA=[xa ya];
XYA=inv(A)*B

%2nd point
xb=0;
yb=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1); k(2)+intd*(sqrt(1+(k(1))^2))];
XYB=[xb yb];
XYB=inv(A)*B

%3rd point
xc=0;
yc=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+intd*(sqrt(1+(c(2))^2)); k(2)+intd*(sqrt(1+(k(1))^2))];
XYC=[xc yc];
XYC=inv(A)*B

%4th point
xd=0;
yd=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+intd*(sqrt(1+(c(2))^2)); k(2)];
XYD=[xd yd];
XYD=inv(A)*B

%5th point
xe=0;
ye=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+intd*(sqrt(1+(c(2))^2)); k(2)-intd*(sqrt(1+(k(1))^2))];
XYE=[xe ye];
XYE=inv(A)*B

%6th point
xf=0;
yf=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1); k(2)-intd*(sqrt(1+(k(1))^2))];
XYF=[xf yf];
XYF=inv(A)*B

%7th point
xg=0;
yg=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-intd*(sqrt(1+(c(2))^2)); k(2)-intd*(sqrt(1+(k(1))^2))];
XYG=[xg yg];
XYG=inv(A)*B
```

```
%8th point
xh=0;
yh=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-intd*(sqrt(1+(c(2))^2)); k(2)];
XYH=[xh yh];
XYH=inv(A)*B

%9th point
xi=0;
yi=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-intd*(sqrt(1+(c(2))^2)); k(2)+intd*(sqrt(1+(k(1))^2))];
XYI=[xi yi];
XYI=inv(A)*B;

%10th point
xj=0;
yj=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)-intd*(sqrt(1+(c(2))^2)); k(2)+2*intd*(sqrt(1+(k(1))^2))];
XYJ=[xj yj];
XYJ=inv(A)*B;

%11th point
xk=0;
yk=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1);  k(2)+2*intd*(sqrt(1+(k(1))^2))];
XYK=[xk yk];
XYK=inv(A)*B;

%12th point
xl=0;
yl=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+intd*(sqrt(1+(c(2))^2)); k(2)+2*intd*(sqrt(1+(k(1))^2))];
XYL=[xl yl];
XYL=inv(A)*B;

%13th point
xm=0;
ym=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+2*intd*(sqrt(1+(c(2))^2)); k(2)+2*intd*(sqrt(1+(k(1))^2))];
XYM=[xm ym];
XYM=inv(A)*B;

%14th point
xn=0;
yn=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+2*intd*(sqrt(1+(c(2))^2)); k(2)+intd*(sqrt(1+(k(1))^2))];
XYN=[xn yn];
```

```
XYN=inv(A)*B;

%15th point
xo=0;
yo=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+2*intd*(sqrt(1+(c(2))^2)); k(2)];
XYO=[xo yo];
XYO=inv(A)*B;

%16th point
xp=0;
yp=0;
A=[1 -c(2); 1 -k(1)];
B=[c(1)+2*intd*(sqrt(1+(c(2))^2)); k(2)-intd*(sqrt(1+(k(1))^2))];
XYP=[xp yp];
XYP=inv(A)*B;

plot(XYA(2),XYA(1),   'bx',   XYB(2),   XYB(1),   'bx',XYC(2),   XYC(1),
'bx',XYD(2), XYD(1), 'bx',...
    XYE(2),   XYE(1),   'bx',XYF(2),   XYF(1),   'bx',XYG(2),   XYG(1),
'bx',XYH(2),  XYH(1),  'bx',XYI(2),  XYI(1),  'bx',  XYJ(2),  XYJ(1),'bx',
XYK(2),XYK(1),'bx',...
    XYL(2),XYL(1),'bx', XYM(2),XYM(1),'bx',XYN(2),XYN(1),  'bx',  XYO(2),
XYO(1),'bx', XYP(2),XYP(1),'bx')

%Distances from the points
for i=1:indexx
da(i)=sqrt((XYA(1)-x_vals(i))^2+(XYA(2)-y_vals(i))^2)
db(i)=sqrt((XYB(1)-x_vals(i))^2+(XYB(2)-y_vals(i))^2)
dc(i)=sqrt((XYC(1)-x_vals(i))^2+(XYC(2)-y_vals(i))^2)
dd(i)=sqrt((XYD(1)-x_vals(i))^2+(XYD(2)-y_vals(i))^2)
de(i)=sqrt((XYE(1)-x_vals(i))^2+(XYE(2)-y_vals(i))^2)
df(i)=sqrt((XYF(1)-x_vals(i))^2+(XYF(2)-y_vals(i))^2)
di(i)=sqrt((XYI(1)-x_vals(i))^2+(XYI(2)-y_vals(i))^2)
dh(i)=sqrt((XYH(1)-x_vals(i))^2+(XYH(2)-y_vals(i))^2)
dj(i)=sqrt((XYJ(1)-x_vals(i))^2+(XYJ(2)-y_vals(i))^2)
dk(i)=sqrt((XYK(1)-x_vals(i))^2+(XYK(2)-y_vals(i))^2)
dl(i)=sqrt((XYL(1)-x_vals(i))^2+(XYL(2)-y_vals(i))^2)
dm(i)=sqrt((XYM(1)-x_vals(i))^2+(XYM(2)-y_vals(i))^2)
dn(i)=sqrt((XYN(1)-x_vals(i))^2+(XYN(2)-y_vals(i))^2)
do(i)=sqrt((XYO(1)-x_vals(i))^2+(XYO(2)-y_vals(i))^2)
dp(i)=sqrt((XYP(1)-x_vals(i))^2+(XYP(2)-y_vals(i))^2)
end

end1=0
vv1=Inf
for i=1:index
    db=sqrt((XYB(2)-x_vals(clusterhead_neighbors(i)))^2+(XYB(1)-
y_vals(clusterhead_neighbors(i)))^2)

    if db<vv1
        vv1=db
        end1=i
    end
end
```

```
vv2=Inf
for i=1:index
    if i~=end1
    dc=sqrt((XYC(2)-x_vals(clusterhead_neighbors(i)))^2+(XYC(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dc<vv2
        vv2=dc
        end2=i
    end
    end
end

vv3=Inf
for i=1:index
    if i~=end1 & i~=end2
    dd=sqrt((XYD(2)-x_vals(clusterhead_neighbors(i)))^2+(XYD(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dd<vv3
        vv3=dd
        end3=i
    end
    end
end
vv4=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3
    de=sqrt((XYE(2)-x_vals(clusterhead_neighbors(i)))^2+(XYE(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if de<vv4
        vv4=de
        end4=i
    end
    end
end

vv5=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4
    df=sqrt((XYF(2)-x_vals(clusterhead_neighbors(i)))^2+(XYF(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if df<vv5
        vv5=df
        end5=i
    end
    end
end

vv6=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5
    dg=sqrt((XYG(2)-x_vals(clusterhead_neighbors(i)))^2+(XYG(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dg<vv6
        vv6=dg
        end6=i
    end
```

```
    end
end

vv7=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6
    dh=sqrt((XYH(2)-x_vals(clusterhead_neighbors(i)))^2+(XYH(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dh<vv7
        vv7=dh
        end7=i
    end
    end
end

vv8=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6
    & i~=end7
    di=sqrt((XYI(2)-x_vals(clusterhead_neighbors(i)))^2+(XYI(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if di<vv8
        vv8=di
        end8=i
    end
    end
end

vv9=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6
       & i~=end7 & i~=end8
    dj=sqrt((XYJ(2)-x_vals(clusterhead_neighbors(i)))^2+(XYJ(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dj<vv9
        vv9=dj
        end9=i
    end
    end
end

vv10=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6
       & i~=end7 & i~=end8 & i~=end9
    dk=sqrt((XYK(2)-x_vals(clusterhead_neighbors(i)))^2+(XYK(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dk<vv10
        vv10=dk
        end10=i
    end
    end
end

vv11=Inf
for i=1:index
```

```
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6 &
       & i~=end7 & i~=end8 & i~=end9 & i~=end10
    dl=sqrt((XYL(2)-x_vals(clusterhead_neighbors(i)))^2+(XYL(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dl<vv11
        vv11=dl
        end11=i
    end
    end
end

vv12=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6 &
             i~=end7 & i~=end8 & i~=end9 & i~=end10 & i~=end11
    dm=sqrt((XYM(2)-x_vals(clusterhead_neighbors(i)))^2+(XYM(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dm<vv12
        vv12=dm
        end12=i
    end
    end
end

vv13=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6 &
i~=end7 & i~=end8 & i~=end9 & i~=end10 & i~=end11 & i~=end12
    dn=sqrt((XYN(2)-x_vals(clusterhead_neighbors(i)))^2+(XYN(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dn<vv13
        vv13=dn
        end13=i
    end
    end
end

vv14=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6 &
         i~=end7 & i~=end8 & i~=end9 & i~=end10 & i~=end11 & i~=end12 &
         i~=end13
    do=sqrt((XYO(2)-x_vals(clusterhead_neighbors(i)))^2+(XYO(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if do<vv14
        vv14=do
        end14=i
    end
    end
end

vv15=Inf
for i=1:index
    if i~=end1 & i~=end2 & i~=end3 & i~=end4 & i~=end5 & i~=end6 &
        i~=end7 & i~=end8 & i~=end9 & i~=end10 & i~=end11 & i~=end12 &
        i~=end13 & i~=end14
```

```
    dp=sqrt((XYP(2)-x_vals(clusterhead_neighbors(i)))^2+(XYP(1)-
y_vals(clusterhead_neighbors(i)))^2)
    if dp<vv15
        vv15=dp
        end15=i
    end
    end
end

d_total1=vv1+vv2+vv3+vv4+vv5+vv6+vv7+vv8+vv9+vv10+vv11+vv12+vv13+vv14+v
v15
%plot the constracted planar array
plot( [ x_vals(clusterhead) x_vals(clusterhead_neighbors(end1)) ] , ...
      [ y_vals(clusterhead)  y_vals(clusterhead_neighbors(end1)) ],'go-
', 'LineWidth',2)
plot(               [               x_vals(clusterhead_neighbors(end1))
x_vals(clusterhead_neighbors(end2)) ] , ...
      [                             y_vals(clusterhead_neighbors(end1))
y_vals(clusterhead_neighbors(end2)) ],'go-', 'LineWidth',2)
plot(               [               x_vals(clusterhead_neighbors(end2))
x_vals(clusterhead_neighbors(end3) )] , ...
      [                             y_vals(clusterhead_neighbors(end2))
y_vals(clusterhead_neighbors(end3)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end3))
x_vals(clusterhead_neighbors(end4) )] , ...
      [                             y_vals(clusterhead_neighbors(end3))
y_vals(clusterhead_neighbors(end4)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end4))
x_vals(clusterhead_neighbors(end5) )] , ...
      [                             y_vals(clusterhead_neighbors(end4))
y_vals(clusterhead_neighbors(end5)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end5))
x_vals(clusterhead_neighbors(end6) )] , ...
      [                             y_vals(clusterhead_neighbors(end5))
y_vals(clusterhead_neighbors(end6)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end6))
x_vals(clusterhead_neighbors(end7) )] , ...
      [                             y_vals(clusterhead_neighbors(end6))
y_vals(clusterhead_neighbors(end7)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end7))
x_vals(clusterhead_neighbors(end8) )] , ...
      [                             y_vals(clusterhead_neighbors(end7))
y_vals(clusterhead_neighbors(end8)) ],'go-', 'LineWidth',2)
   plot(             [               x_vals(clusterhead_neighbors(end8))
x_vals(clusterhead_neighbors(end9) )] , ...
      [                             y_vals(clusterhead_neighbors(end8))
y_vals(clusterhead_neighbors(end9)) ],'go-', 'LineWidth',2)
  plot(             [               x_vals(clusterhead_neighbors(end9))
x_vals(clusterhead_neighbors(end10) )] , ...
      [                             y_vals(clusterhead_neighbors(end9))
y_vals(clusterhead_neighbors(end10)) ],'go-', 'LineWidth',2)
  plot(             [              x_vals(clusterhead_neighbors(end10))
x_vals(clusterhead_neighbors(end11) )] , ...
      [                            y_vals(clusterhead_neighbors(end10))
y_vals(clusterhead_neighbors(end11)) ],'go-', 'LineWidth',2)
  plot(             [              x_vals(clusterhead_neighbors(end11))
x_vals(clusterhead_neighbors(end12) )] , ...
```

```
      [                              y_vals(clusterhead_neighbors(end11))
y_vals(clusterhead_neighbors(end12)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end12))
x_vals(clusterhead_neighbors(end13) )] , ...
      [                              y_vals(clusterhead_neighbors(end12))
y_vals(clusterhead_neighbors(end13)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end13))
x_vals(clusterhead_neighbors(end14) )] , ...
      [                              y_vals(clusterhead_neighbors(end13))
y_vals(clusterhead_neighbors(end14)) ],'go-', 'LineWidth',2)
  plot(              [              x_vals(clusterhead_neighbors(end14))
x_vals(clusterhead_neighbors(end15) )] , ...
      [                              y_vals(clusterhead_neighbors(end14))
y_vals(clusterhead_neighbors(end15)) ],'go-', 'LineWidth',2)
```

```
%beamforming function
clear all
close all
clc

global c f l b Im Nx Ny
c=3e8;
f=0.3e9;
l=c/f;
b=2*pi/l;

GdBlsavg=zeros(181,181);
X300
Y300

NumIter=10;
for i=1:NumIter;
    XX=X300(i,:);
    YY=Y300(i,:);
    XX=XX';
    YY=YY';
XXX=[ones(7,1) XX];
YYY=YY;
C(:,i)=XXX\YYY;
end

F=C(2,:);
FF=F.*(180/pi);

%%%% X,Y  matrix 100x7
NumIter=10;

for i=1:NumIter;
    x=X300(i,:)';
    y=Y300(i,:)';

    x=x-x(1);
    y=y-y(1);

Nx=7   ;      % Nx number of array elements in x direction
Ny=1   ;      % Ny number of array elements in y direction

theta0=0 ;     % Elevation angle theta (degrees)
theta0=theta0*pi/180;

phi0=FF(i);       % Azimuth angle  phi (degrees)
phi0=phi0*pi/180;
phi_ang=FF(i);   % Angle phi for beampattern

phi00=0;
phi00=phi00*pi/180;
phi00_ang=phi00;
```

```
%%%%%%%%%    Uniform array (reference)  %%%%%%%%%%
dx=l/2; % ideal distance ?/2 in x-direction
xn=(0:Nx-1)*dx;
xn=repmat(xn',1,Ny);
xn=reshape(xn,Nx*Ny,1);

dy=l/2;  % ideal distance ?/2 in y-direction
yn=(0:Ny-1)*dy;
yn=repmat(yn,Nx,1);
yn=reshape(yn,Nx*Ny,1);

%%%%%%%%%    End of uniform array   %%%%%%%%%%



%%%%%%%%%    Beampattern Uniform Array
Im=ones(Nx*Ny,1);    % Amplitudes
wref=weights2(xn,yn,theta0,phi00);      %  Reference  weights  (uniform
array)
Gref=gain2D(wref,xn,yn);
GdBref=10*log10(Gref/max(max(Gref)));   %  Gain  for  reference  (uniform
array)

%%%%%%%%% Beampattern with errors - LS solution
theta=-90:90;
th=theta*pi/180;

dn=exp(j*b*(xn*sin(th)*cos(phi00)+yn*sin(th)*sin(phi00)));   %  steering
vector for ULA
Fdes=wref'*dn;
d=exp(j*b*(x*sin(th)*cos(phi0)+y*sin(th)*sin(phi0)));      %   steering
vector

ww=Fdes/d;
ww=ww';   %  or ww=inv(d*d')*d*Fdes';

Gls=gain2D(ww,x,y);
GdBls=10*log10(Gls/max(max(Gref)));
GdBlsavg=GdBlsavg+GdBls;
end

GdBlsavg=GdBlsavg/NumIter;

figure(3);
theta=-90:90;
plot(theta,GdBref(:,phi00_ang+90+1),'b-','Linewidth',2);
hold on;
plot(theta,GdBlsavg(:,phi00_ang+90+1),'r--','Linewidth',1);
grid on;
legend('Ideal Linear','Actual Positions');
axis([-85 85 -50 5]);
title('Fig.1 : Average Beampattern for 7 linear array elements and
given \phi','Fontsize',12);
xlabel('\theta (degrees)','Fontsize',12);
ylabel('Power Gain (dB),' ,'Fontsize',12);
```

LIST OF REFERENCES

[1] P.J. Vincent, M. Tummala and J. McEachen, "An Energy-Efficient Approach for Information Transfer from Distributed Wireless Sensor Systems," *Proceeding of the 2006 IEEE International Conference on System of Systems Engineering,* Los Angeles, CA, USA, pp. 100-105, April 2006.

[2] M. Tummala, C.C. Wai and P. Vincent, "Distributed Beamforming in Wireless Sensor Networks," *Proceedings of the 39th Asilomar Conference on Signals, Systems and Computers*, pp 793-797, October-November 2005.

[3] P.J. Vincent, M. Tummala and J. McEachen, "A New Method for Distributing Power Usage across a Sensor Network," *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, SECON'06*,Vol 2, pp. 518-526, September 2006.

[4] J. Lee, S. Venkatesh, and M. Kumar, " Formation of a geometric pattern with a mobile wireless sensor network." *J. Robot. Syst.* 21, 10 (Oct. 2004), 517-530.

[5] G. Elissaios and A. Manikas, "Array Formation in Arrayed Wireless Sensor Networks," Communications and Signal Processing Research Group," *HERMIS-mu-pi International Journal of Computer Mathematics and its Applications*, vol.6, pp.122-134, March 2006.

[6] H. Ochiai et al., "Collaborative Beamforming for Distributed Wireless Ad Hoc Sensor Networks," *IEEE Trans. on Signal Processing*, Vol 53, pp 4110-4124, March 2005.

[7] R. Shorey et al., *Mobile, Wireless, and Sensor Networks, Technology, Applications, and Future Directions*, IEEE Press & WILEY-INTERSCIENCE, 2006.

[8] I. Akyildiz et al., "A Survey on Sensor Networks,", *IEEE Communications Magazine*, Vol 40, pp 102-114, August 2002.

[9] N. Bulusu et al., "Scalable Coordination for Wireless Sensor Networks: Self-Configuring Localization Systems," *ISCTA 2001*, Ambleside, UK, July 2001.

[10] D. Estrin et al., "Next Century Challenges: Scalable Coordination in Sensor Networks,", *In Mobile Computing and Networking*, pp 263-270, 1999.

[11] M. Yarvis and W. Ye, *Tiered Architectures in Sensor Networks*, Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, Ch. 13, CRC Press, 2005.

[12]] E. Shlh et al., "Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks," *Proc. ACM MobiCom '01*, pp272-86, Rome-Italy, July 2001.

[13] W.L. Stutzman and A. Garry. *Thiele, Antenna Theory and Design*, John Wiley & Sons, NY, 1998.

[14] United States Naval Academy, "Uniform Random Area Search", http://www.nadn.navy.mil/MathDept/courses/pre97/sm230/urs.htm, Last Accessed May 19, 2007.

[15] C. C. Wai, "Distributed Beamforming in Wireless Sensor Networks," Master's Thesis, Naval Postgraduate School, Monterey, CA, December 2004.

[16] S. Leon, *Linear Algebra with applications*, Prentice Hall, 6th edition, 2002.

[17] N. Papalexidis, "Distributed Beamforming in Wireless Sensor Networks," EE and Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2007.

[18] M. I. Skolnik, *Introduction to Radar Systems*, 3rd edition, McGrow-Hill, New York, NY, 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Hellenic Navy General Staff
   Department B2
   Athens, Greece

4. Professor Jeffrey Knorr, Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Professor Murali Tummala
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Professor John C. McEachen
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

7. CDR T. Owens Walker III
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

8. LT Charalampos Gkionis HN
   Hellenic Navy General Staff
   Athens, Greece

9. Mike Niermann
   SPAWAR
   Charleston, South Carolina

10. Martin Kruger
    ONR
    Arlington, Virginia

11. Bernie Schneider
SOCOM
McDill AFB, Florida

12. Jamie Carson
SRC
Charleston, South Carolina

13. George Hinckley
Virginia Advanced R&D Initiative
Quantico, Virginia

14. Richard Wylly
SRC
Charleston, South Carolina