NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DISTRIBUTED ALGORITHMS FOR BEAMFORMING IN WIRLESS SENSOR NETWORKS

by

Nikolaos Papalexidis

June 2007

Thesis Advisor:	Murali Tummala
Thesis Co-Advisor:	John C. McEachen
Thesis Committee Members:	Roberto Cristi
	Weilian Su

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2007	Engineer's Thesis

4. TITLE AND SUBTITLE Distributed Algorithms for Beamforming in Wireless Sensor Networks	5. FUNDING NUMBERS
6. AUTHOR(S) Nikolaos Papalexidis	
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	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT (maximum 200 words)

Sensor nodes in a wireless sensor network (WSN) can establish a link with a UAV by using beamforming techniques to form a random array with position errors. The position errors' effect in the array performance is examined using a MATLAB-based simulation model.

In order to spread the processing and communication load among the nodes, two new distributed algorithms for beamforming in WSN, based on the least squares (LS) approximation of the desired array response, are proposed. The first is a distributed implementation of the QR decomposition, and the second is an iterative method for solving the LS problem. Results indicate that the processing load is effectively shared among the nodes. Especially, in the second approach, the processing load can be lower than that of the centralized approach, depending on the algorithm's convergence. For both algorithms, the tradeoff for the ability to spread the processing load is the increased communication cost, which could cause an overall increase in the total power consumption in the network. However, the average power per participating sensor node is still lower than that required by the cluster head in the centralized approach. Consequently, the network's susceptibility to failures due to excessive power consumption is greatly reduced.

14. SUBJECT TERMS wireless sensor networks, distributed beamforming, distributed QR decomposition, iterative least squares	15. NUMBER OF PAGES 136
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	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 release; distribution is unlimited

DISTRIBUTED ALGORITHMS FOR BEAMFORMING IN WIRELESS SENSOR NETWORKS

Nikolaos Papalexidis
Lieutenant, Hellenic Air Force
B.S., Hellenic Air Force, Athens, 1999

Submitted in partial fulfillment of the
requirements for the degrees of

ELECTRICAL ENGINEER

and

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author: Nikolaos Papalexidis

Approved by: Murali Tummala
Thesis Advisor

John C. McEachen
Thesis Co-Advisor

Roberto Cristi
Thesis Committee Member

Weilian Su
Thesis Committee Member

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Sensor nodes in a wireless sensor network (WSN) can establish a link with a UAV by using beamforming techniques to form a random array with position errors. The position errors' effect in the array performance is examined using a MATLAB-based simulation model.

In order to spread the processing and communication load among the nodes, two new distributed algorithms for beamforming in WSN, based on the least squares (LS) approximation of the desired array response, are proposed. The first is a distributed implementation of the QR decomposition, and the second is an iterative method for solving the LS problem. Results indicate that the processing load is effectively shared among the nodes. Especially, in the second approach, the processing load can be lower than that of the centralized approach, depending on the algorithm's convergence. For both algorithms, the tradeoff for the ability to spread the processing load is the increased communication cost, which could cause an overall increase in the total power consumption in the network. However, the average power per participating sensor node is still lower than that required by the cluster head in the centralized approach. Consequently, the network's susceptibility to failures due to excessive power consumption is greatly reduced.

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 gratitude to Professor Murali Tummala for his support, guidance and motivation throughout this work. I am grateful for his support in the pursuit of my interests. It has been a real privilege to work with you.

I would like to thank CDR T. Owens Walker for his valuable contribution throughout this research.

I also express my great appreciation to Professors John McEachen, Roberto Cristi and Weilian Su for their assistance and recommendations for improving this work.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

A wireless sensor network (WSN) consists of a large number of microsensors, each having limited battery lifetime and restricted communication and computing capabilities. Recent advances in the integrated circuit technology have allowed the production of lightweight and inexpensive sensor nodes, which have a range of capabilities, such as sensing, processing and communication. WSNs can have many applications in both commercial and military environments. A set of sensor nodes, which can be deployed easily and quickly by an unmanned aerial vehicle (UAV), for example, can be used for monitoring the battlefield environment, sensing for a wide range of targets, especially in the case where the area of interest is inaccessible or there is high risk of human loss.

Once deployed, the sensor nodes can collect the desired information and transmit it to the UAV. Although a single sensor node cannot transmit its data directly to the UAV due to the limited range of coverage, several of them can coordinate their transmissions in order to form an array and thus substantially increase the range of coverage. Since the sensors are randomly deployed, it is unlikely that they form topologies that permit the formation of arrays with equally spaced elements. As a result, there are position errors with respect to an array with equally spaced elements.

A simulation model was developed in the MATLAB environment to analyze the effect of these position errors on the array performance. The simulations showed that the sidelobe levels in the array response increase as a function of the error in the element location. Specifically, as the mean deviation from the ideal position was increased, the average sidelobe magnitudes also increased. These results are in agreement with the theoretical analysis of random arrays, found in the literature. The degradation of the array performance can be largely eliminated using a Least Squares (LS) beamformer, which computes the weights that best approximate a given desired response. This beamformer can also efficiently suppress potential interfering signals coming from directions other than the signal's.

Since reliability and robustness in a sensor network environment are desired, the processing load must be effectively spread among the sensor nodes. Centralized approaches assign the entire processing load to a single node whereas in a distributed approach, the processing tasks are split into smaller processes, which are then allocated to the participating sensor nodes. Two fully distributed approaches to beamforming in WSN were presented in this work, and they are both based on the LS approximation of the desired response. The first is a distributed implementation of the QR decomposition, and the second is an iterative method of computing the weights in the LS sense.

The performance of the distributed methods was compared to the centralized LS approach using the processing and communication costs as metrics. The results indicate that the processing load is effectively shared among the nodes. Especially in the second method, the processing load is a function of the algorithm's convergence and can be lower compared to that of the centralized approach, subject to the speed of convergence. For both algorithms, the tradeoff for the ability to spread the processing load is the increased communication cost, which could cause an overall increase in the total power consumption in the network. This total power, however, is shared among the sensor nodes; therefore, the average power expended by a participating sensor node in the distributed implementation is lower than the power required by the cluster head in the centralized approach. Consequently, the network's susceptibility to failures due to excessive power consumption is greatly reduced.

I. INTRODUCTION

A. INTRODUCTION TO WIRELESS SENSOR NETWORKS

A wireless sensor network (WSN) consists of a large number of microsensors, each having limited battery lifetime and, therefore, restricted communication and computing capabilities [1], [2] . Recent advances in the integrated circuit technology have allowed the production of lightweight and inexpensive sensor nodes, which have a range of capabilities, such as sensing, processing and communication. If they are properly networked and programmed, these sensor nodes can cooperate in order to perform complex signal processing functions [3], [4].

The main issue for a WSN is to prolong its operational lifetime as much as possible, taking into account the sensors' power consumption requirements. Stringent energy limitations are also a crucial factor when designing signal processing algorithms for a WSN [5]. Since such energy restrictions are not taken into account by the signal processing methods that are already used in applications other than WSNs, existing techniques should be modified in order to conform to the sensor nodes' specific characteristics. Therefore, a major challenge in recent research is the design of signal processing and networking operations, which optimize the tradeoff between energy efficiency, simplicity, and overall performance, [3].

Because microsensors are becoming cheaper and more capable, WSNs will find more applications in both commercial and military environments [1], [2]. Future tactical operations will involve the deployment of large-scale WSNs in which hundreds or thousands of disposable sensor nodes will cooperate in order to achieve the mission objective [3]. These nodes can be deployed easily and quickly by an unmanned aerial vehicle (UAV), for example, as in Figure 1, which minimizes the risk of human loss. Then they can be used for monitoring the battlefield environment, sensing for a wide range of targets, such as biological, radioactive, nuclear, chemical and other materials [1]. Once deployed, the sensor nodes can collect the desired information and disseminate it to a relay node, such as a UAV. Furthermore, taking into account that the sensing

environment may be harsh and inaccessible for deploying wired networks, there is obvious need for developing WSNs consisting of small and disposable sensors.

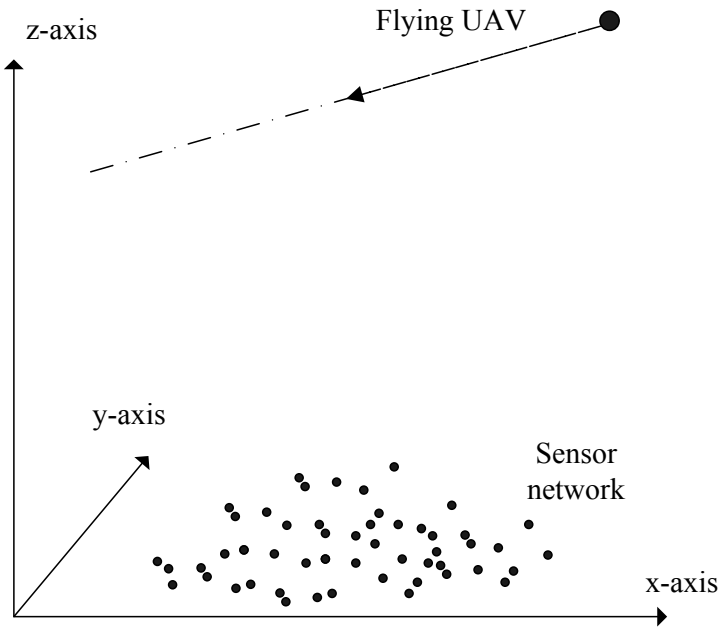


Figure 1. WSN deployed over an area of interest and UAV collecting the desired information.

B. RELATED WORK IN BEAMFORMING AND WIRELESS SENSOR NETWORKS

After forming an ad hoc network and collecting the required data about the target of interest, the sensor nodes must establish communication with a UAV, so the acquired information can be transmitted to the UAV. Although promising, today's technology still imposes strict limits on the processing and communication capabilities of the sensor nodes [6], [7]. Single sensor nodes do not have sufficient power to communicate with an overflying UAV. Since the UAV may be required to fly at a high altitude due to the hostile nature of the operative environment, the objective of transmitting the collected information to the UAV becomes more difficult.

Although a single sensor node cannot transmit its data directly to the UAV due to the limited batter power, several of them can cooperate in order to function as a large antenna array and thus substantially increase the transmission range and the data rate.

This process of combining the signal from different antenna elements in order to form a single output of the sensor array is known as beamforming. It has been proven by many studies that when an antenna array is properly configured, it can improve the channel capacity and extend the range of coverage [8], [9]. It can also reduce the multipath fading and the bit error rate (BER); therefore, it results in more reliable communication [8]. Additionally, beamforming can adaptively steer the antenna beam towards the UAV, thus aiming the radiated energy in the desired direction [10]. Furthermore, the antenna gain is proportional to the number of the antenna elements, so the main beam peak power density can be of several orders of magnitude higher than that of a single sensor [9]. Another useful characteristic of an antenna array is that it can be used in order to perform spatio-temporal filtering, thus suppressing potential interference signals coming from directions other than the desired direction [11], [12]. In summary, taking the above mentioned advantages into account, beamforming in WSNs can meet the objective of establishing an efficient communication link between a WSN and a UAV.

Several algorithms for beamforming exist in the literature [13], [11], [14] and many of them are successfully implemented in conventional antenna arrays. [10]. Nevertheless, these algorithms for the computation of the weights for the array elements cannot be directly implemented in WSNs since there are significant differences between WSNs and conventional arrays. For instance, the phased arrays used in RADAR are installed permanently on site [15]; thus, the positions of the antenna elements are fixed. On the other hand, the sensor nodes in a WSN are usually randomly deployed and their relative positions are not predetermined. Due to this random deployment, there are position errors, which cause performance deterioration of the antenna array, compared to that of an array of equally spaced elements. Moreover, the sensors are prone to frequent failures due to limited battery life or due to their vulnerability to environmental conditions. Therefore, the topology of the sensor array can change substantially as new nodes are added or withdrawn.

Another significant problem is that, in conventional arrays, where power is not a major issue, the beamforming operation is performed in a single processor [16]. All necessary information is collected in a central processing node, which is responsible for

solving the beamforming problem. However, in a sensor network environment, this is neither reliable nor desirable since a single node would be assigned this computationally demanding task. Additionally, such centralized implementations create a single point of failure, which in turn creates a serious system vulnerability. If this node fails, then the beamforming problem has to be solved from the beginning. Thus, the processing load or consequently the power usage should be effectively distributed across the sensor network [17]. However, an optimum set of participating sensors in the array has to be defined since the communication cost for organizing the sensor nodes into an array is prohibitive after a certain critical number of nodes [18].

C. THESIS OBJECTIVE

The objective of this research is to implement several distributed beamforming algorithms, and to evaluate their performance. Throughout this work, the operational scenario of Figure 1 is adopted where several sensor nodes try to communicate with a UAV. The beamforming process is not performed in a central node (cluster head), but it is split into smaller processes, which then can be allocated to the sensor nodes. The main concept is that the processing and communication cost must be shared among the nodes, so there is no single point of failure and that the energy of the nodes is efficiently used, thus extending the WSN lifetime. This work is focused on investigating beamforming schemes that have the same performance as well established centralized approaches yet offer the advantage of implementation in a distributed fashion thus, increasing the network's robustness and overall performance.

D. PROPOSED APPROACH TO DISTRIBUTED BEAMFORMING

Starting from a centralized approach to the Least Squares (LS) solution of the beamforming problem where the beamformer is designed in such way that the desired array performance is best approximated, two fully distributed methods are proposed. The first one is a distributed implementation of the QR decomposition with Householder

transformations and derives the exact solution for the array weight vector. The second scheme is an iterative method for solving the LS problem, implemented in a distributed fashion.

In order to examine the proposed techniques, a performance analysis of the communication and computational costs is developed. These two costs are closely connected to the power consumption and provide a reliable test for the algorithms' effectiveness. The resulting array response is examined and compared with the desired response. The results from these two distributed implementations are encouraging and indicate that they can provide a realistic solution for the beamforming problem in sensor networks.

E. THESIS OUTLINE

Chapter II introduces the fundamental concepts of the antenna arrays, including a description of the uniform linear and planar array. This is followed by an analysis of the effects of position errors on the performance of the antenna array and a simulation, which confirms the theoretical results. Beamforming in wireless sensor networks is also presented along with a specific operational communication scenario which uses a UAV. Finally, the framework for evaluating the algorithms' performance based on the factors that affect power consumption, such as the processing and communication cost, is developed.

Chapter III presents two centralized beamforming approaches and evaluates their performance. Their advantages and disadvantages are discussed, and their ability to mitigate position errors is analyzed using a simulation model developed in a MATLAB environment.

In Chapter IV, two distributed algorithms for beamforming in wireless sensor networks are proposed. Their performance is analyzed in terms of efficiently sharing the processing and communication cost among the nodes, and the simulation results are compared to those obtained by the centralized approaches.

Chapter V summarizes the significant results of this thesis and provides some ideas for extending this work in the future.

Finally, the Appendix includes the MATLAB code used in the simulations.

II. BEAMFORMING IN SENSOR ARRAYS

In this chapter, the main concepts of antenna arrays are discussed; specifically, the uniform linear and planar arrays are presented as well as their array response (beampattern). The uniform array is followed by an analysis of the random array with position errors. The effects of the antenna element position errors in the main lobe and the sidelobe power gain are presented under various assumptions about the statistical characteristics of the error. Beamformer implementations are described, particularly the narrowband beamformer. Lastly, there is a discussion about communication and computational cost in sensor networks as a function of power consumption, which indicates the need for distributed algorithms in beamforming.

A. UNIFORM LINEAR AND PLANAR ARRAYS

The fundamental concepts of the uniform linear and planar arrays are presented in this section along with the basic formulation of beamforming which will be used throughout this work.

1. One-Dimensional Array

In Figure 2, a linear array is depicted with M identical, equally spaced elements. The spacing between consecutive array elements or sensor nodes in the case of a sensor array is assumed equal to a half wavelength, i.e., $d = \lambda / 2$, where the elements are isotropic, meaning their beampattern is omni-directional. Furthermore, it is assumed that the array is located far enough from the signal source (e.g., a UAV); thus, it is considered to lie in the source's far field. The array axis is assumed to be in the x-axis. The plane wave $s(t)$ arrives at the array at an angle θ_a with respect to the x-axis (array axis), and for this reason, the $m+1^{th}$ element senses the signal earlier than the m^{th} element. If x_m is the distance between the m^{th} node and the reference node, which is located at the origin of the coordinate system, then the signal $s(t)$ arrives at the m^{th} element earlier by t_m

seconds with respect to the reference element. The time difference, which depends on the arrival angle θ_{α_0} or Angle of Arrival (AOA), and the element's distance from the reference point is given by [19]:

$$t_m\left(\theta_{a_0}\right) = \frac{x_m \cos\theta_{a_0}}{c} \tag{1}$$

where c is the speed of light.

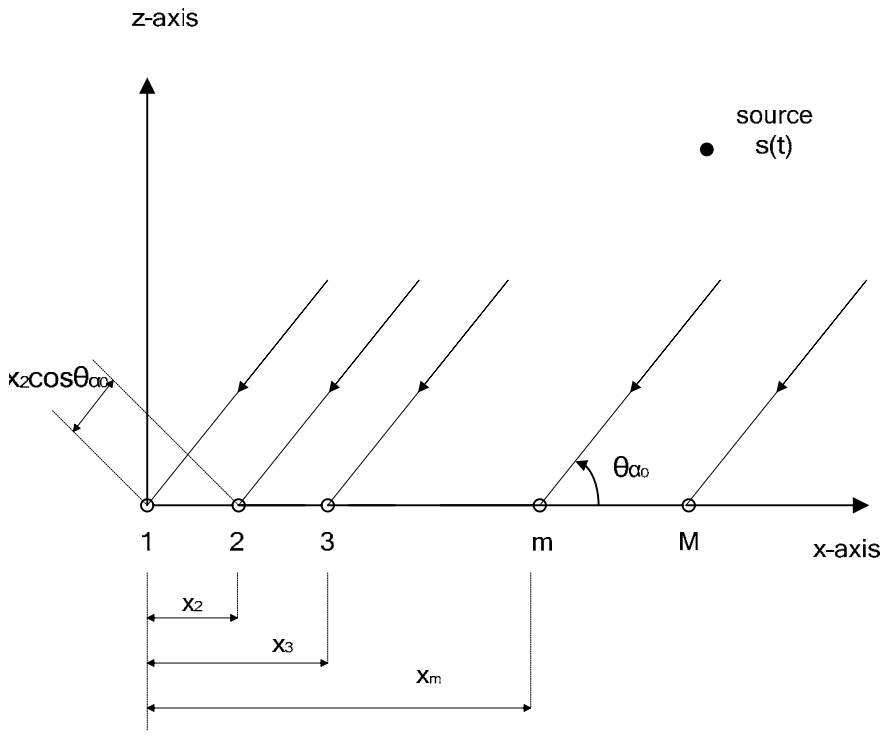


Figure 2. An $M \times 1$ linear array of equally spaced isotropic elements.

Each array element is weighted by a complex weight w_m, for $m = 0,1,\ldots,M$, which multiplies the incoming signal. Adding all the elements' weighted inputs gives the spatial response of the array or array factor $F(\theta_a)$ for any arbitrary angle θ_a:

$$F(\theta_a) = \sum_{m=1}^{M} w_m^* e^{j\omega t_m(\theta_a)} \tag{2}$$

where $w_m = I_m e^{j\omega t_m \theta_{a_0}}$ and I_m and $e^{j\omega t_m \theta_{a_0}}$ are the magnitude and the phase of the

complex weights, respectively. Using the wavenumber $\beta = 2\pi / \lambda$ and the expression for the time difference t_m, the weights can be written as $w_m = I_m e^{j\beta x_m \cos\theta_{a_0}}$ and the array factor as

$$F(\theta_a) = \sum_{m=1}^{M} w_m^* e^{j\beta x_m \cos\theta_a} \tag{3}$$

The weights w_m are carefully selected in order to give the maximum value of the array response $F(\theta_a)$ at the desired direction θ_{a_0} and to suppress potential interference signals arriving from other directions. Indeed at $\theta_a = \theta_{a_0}$, the array response reaches its maximum value

$$F(\theta_{a_0}) = \sum_{m=1}^{M} I_m e^{-j\beta x_m \cos\theta_{a_0}} e^{j\beta x_m \cos\theta_{a_0}} = \sum_{m=1}^{M} I_m = M \qquad \text{if} \quad I_m = 1, \forall m \tag{4}$$

The set of weights w_m form the weight vector

$$\underline{w} = [w_1 \quad w_2 \quad \ldots \quad w_m]^T$$

while the steering vector or direction vector is defined as

$$\underline{d}(\theta_a) = [1 \quad e^{j\beta x_2 \cos\theta_a} \quad e^{j\beta x_3 \cos\theta_a} \quad \ldots \quad e^{j\beta x_M \cos\theta_a}]^T$$

and incorporates the location information of the array. Therefore, the array response can be expressed as

$$F(\theta_a) = \underline{w}^H \underline{d}(\theta_a). \tag{5}$$

Note that the main concept of beamforming is the use of the weights $\underline{w}$ in order to point the array beam towards any desired direction. So, if the desired transmission direction is θ_{a_0}, then the beamformer should set its weights to be

$$w_m = I_m e^{j\beta x_m \cos\theta_{a_o}}. \tag{6}$$

The beampattern of a 10×1 uniform linear array is shown in Figure 3 where the normalized power gain G is defined as [9]

$$G(\theta_a) = \frac{|F(\theta_a)|^2}{\max\limits_{\theta_a} |F(\theta_a)|^2} \tag{7}$$

and it is plotted as a function of the direction θ_a (in degrees). The array elements are identical and isotropic, and the spacing has a fixed value of $\lambda/2$ and beam pointing angle $\theta_{a_0} = 0°$. The maximum sidelobe is equal to -13 dB and the Half Power Beamwidth (HPBW) is about $10.2°$.

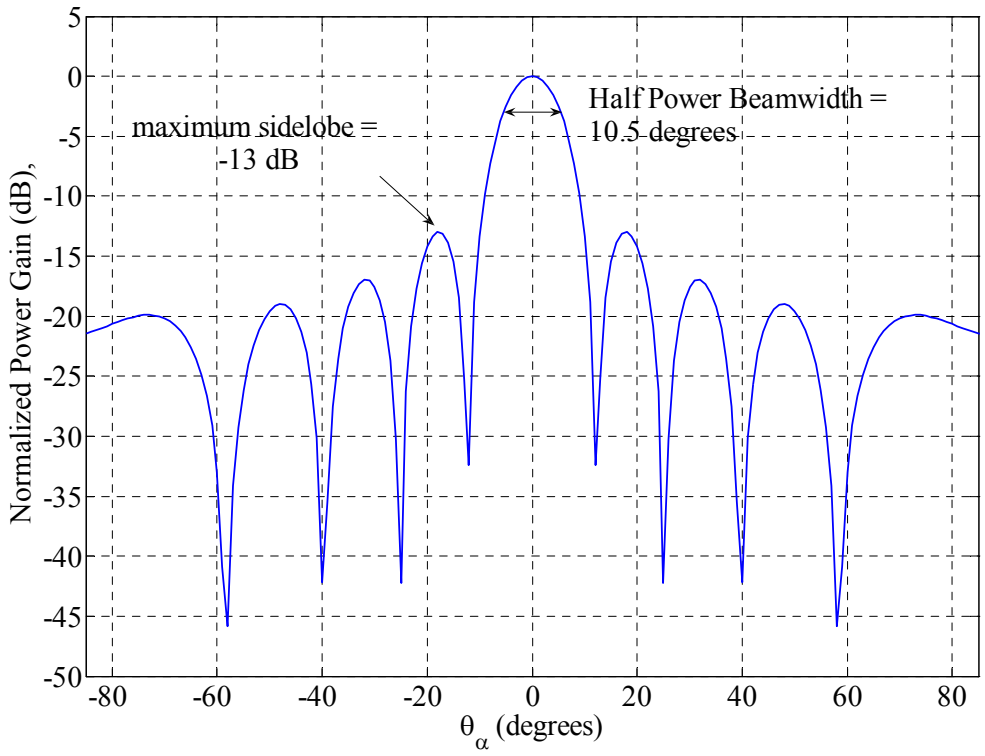


Figure 3. Normalized Power Gain (dB) – Beampattern of a $M = 10$ element array with isotropic elements, fixed spacing $\lambda/2$ and $\theta_{a_0} = 0°$.

In general, the sidelobe level decreases with increasing number of elements M and approaches the value of -13.3 dB. The HPBW in any plane containing the array axis. is given [9] by

$$\theta_{3dB} = 0.866 \frac{\lambda}{Md}$$

where $Md \gg \lambda$, i.e., it is valid for long arrays. Therefore the HPBW is a function of M and decreases as the number of array elements increase.

2. Two-dimensional (planar) Array

The previous discussion of a linear array can be easily expanded to a planar array, and similar expressions can be derived for the array response and the power gain. In Figure 4, a uniformly spaced planar array is depicted with $M \times N$ identical and isotropic elements. The spacing between the array elements is assumed equal to half wavelength, $d = \lambda/2$, in both directions while it is assumed again that the array is located in the source's far field. The plane wave $s(t)$ arrives at the array at polar angle θ_0 with respect to the z-axis and an azimuth angle ϕ_o with respect to the x-axis; thus, the $(m,n)^{th}$ element receives the signal earlier by t_{mn} seconds compared to the reference element at the origin. This time difference t_{mn} depends on the angles θ_0, ϕ_o and the element's position (x_{mn}, y_{mn}) in the array, and is given by [19]

$$t_{mn}(\theta_0, \phi_0) = \frac{x_{mn} \sin\theta_0 \cos\phi_0 + y_{mn} \sin\theta_0 \sin\phi_0}{c} \tag{8}$$

Adding all the elements' weighted inputs gives the array response of the planar array $F(\theta,\phi)$ for any arbitrary choice of angles θ and ϕ :

$$F(\theta,\phi) = \sum_{n=1}^{N} \sum_{m=1}^{M} w_{mn}^{*} e^{j\beta(x_{mn}\sin\theta\cos\phi + y_{mn}\sin\theta\sin\phi)} \tag{9}$$

where

$$w_{mn} = I_{mn} e^{j\beta(x_{mn}\sin\theta_0\cos\phi_0 + y_{mn}\sin\theta_0\sin\phi_0)} \tag{10}$$

are the complex weights for each element (m,n). In matrix form, the above expression can be written as

$$F(\theta,\phi) = \underline{w}^{H}\, \underline{d}(\theta,\phi) \tag{11}$$

where $\underline{w}$ is an $MN \times 1$ weight vector and

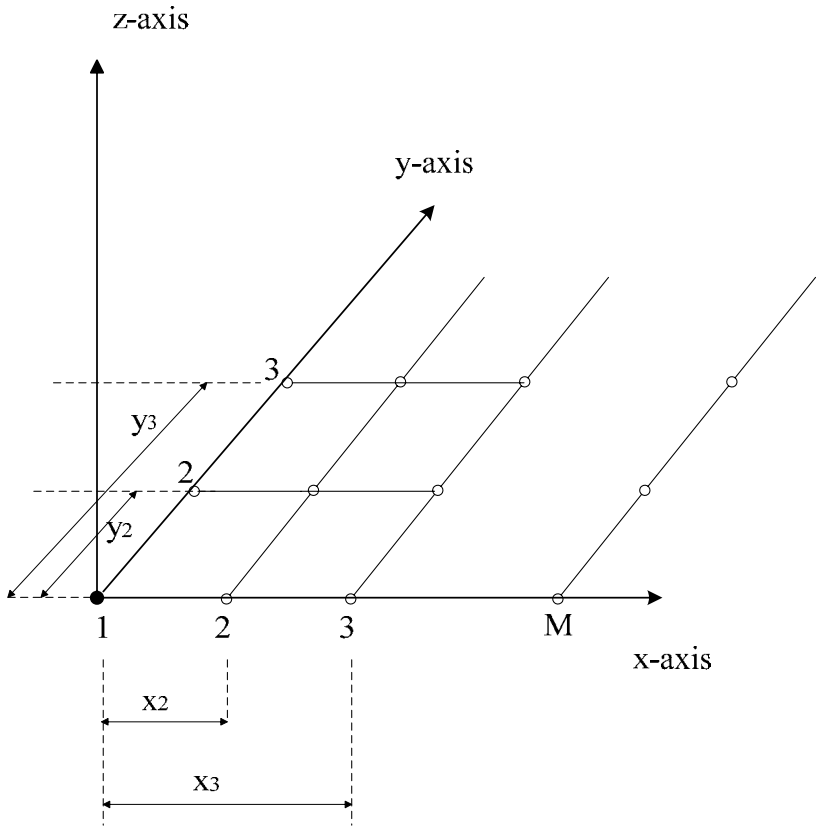
$$\underline{d}(\theta,\phi)=\begin{bmatrix} 1 \\ e^{j\beta(x_{12}\sin\theta\cos\phi+y_{12}\sin\theta\sin\phi)} \\ e^{j\beta(x_{13}\sin\theta\cos\phi+y_{13}\sin\theta\sin\phi)} \\ \vdots \\ e^{j\beta(x_{1M}\sin\theta\cos\phi+y_{1M}\sin\theta\sin\phi)} \\ \vdots \\ e^{j\beta(x_{MN}\sin\theta\cos\phi+y_{MN}\sin\theta\sin\phi)} \end{bmatrix} \tag{12}$$

is an $MN\times 1$ steering vector.

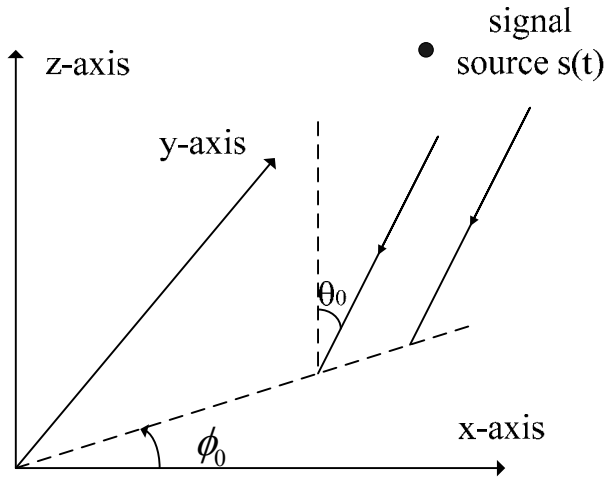
The weights w_{mn} are again selected in order to give the maximum value of the array response $F(\theta,\varphi)$ in the desired direction $(\theta_{0,}\phi_0)$. At $\theta=\theta_0$ and $\phi=\phi_0$, the array response reaches its maximum value

$$F(\theta_0,\varphi_0)=MN \qquad \text{if } I_{mn}=1,\forall m,n \tag{13}$$

where in (13) it is assumed that the excitation of elements (weighting) is uniform.



a) $M \times N$ planar array of equally spaced isotropic elements



b) Measurement of angles for direction of signal arrival (θ_0, ϕ_0)

Figure 4. Signal wavefront arriving from direction (θ_0, ϕ_0) at a $M \times N$ planar array.

In Figure 5, the three-dimensional beampattern of a 10×10 uniform planar array is shown, where the normalized power gain G in dB is plotted as a function of the polar angle θ and the azimuth angle ϕ. A cross section of this 3-D beampattern is plotted in Figure 6 where the azimuth angle is constant at $\phi_0 = 45°$ (direction of arrival), and the beampattern varies with the polar angle θ.

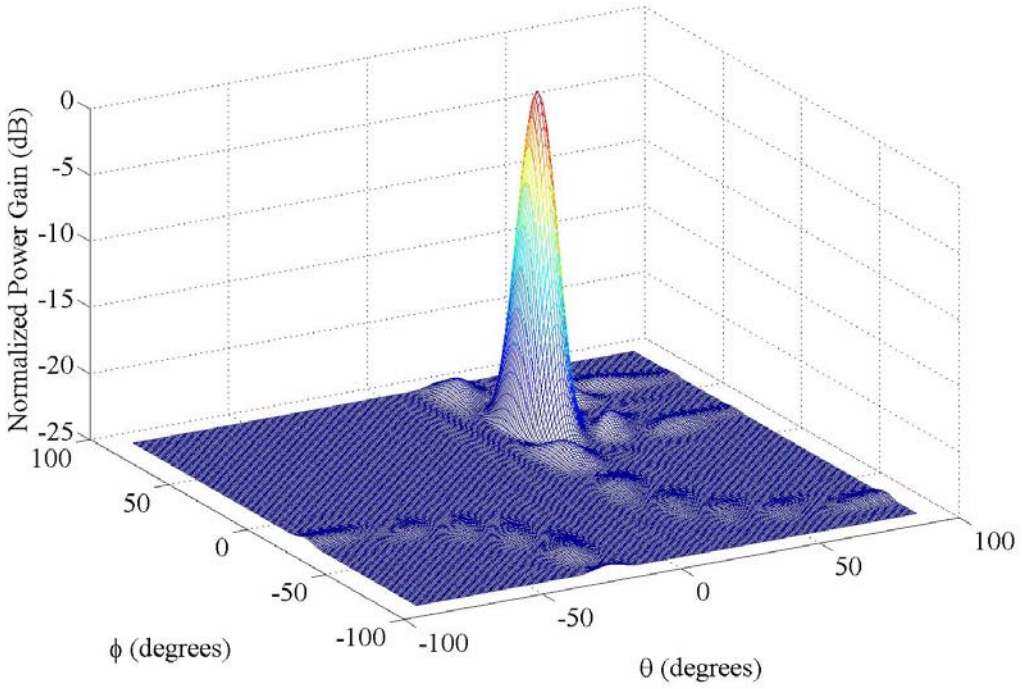


Figure 5. Normalized Power Gain (dB): 3-D Beampattern of a 10×10 planar array with isotropic elements, equally spaced by $\lambda/2$ and direction of signal arrival $(\theta_0 = 30°, \phi_0 = 45°)$.

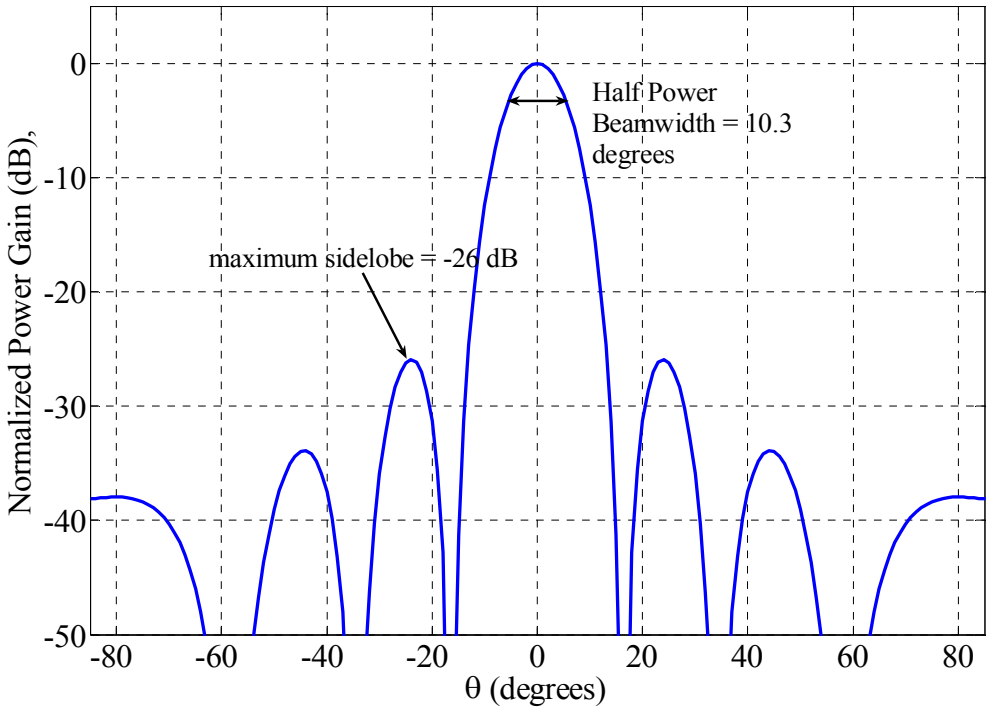


Figure 6. Normalized Power Gain (dB): Beampattern of a 10×10 planar array with isotropic elements, equally spaced by $\lambda/2$ and direction of signal arrival $(\theta_0 = 0°, \phi_0 = 45°)$. Azimuth angle is fixed at $\phi_0 = 45°$.

The maximum sidelobe is equal to -26 dB, and the HPBW is about $10.2°$. In general, the sidelobe levels decrease with an increasing number of elements MN and approaches the value of -26.6 dB. Similarly, the HPBW decreases continuously as the number of elements increase.

B. RANDOM ARRAY: POSITION ERRORS

In a random deployment of a sensor array where the sensor nodes are dropped randomly over an area of interest, it would be unrealistic to expect formations of perfectly spaced planar arrays. An illustrating example comes from the fact that for an operating frequency f_c = 1 GHz, the ideal distance between the sensor nodes is $\lambda/2 = 15$ cm, and even a small displacement of 3 cm yields a position error of 20%. Therefore, the performance of a sensor array should be studied using the theoretical

analysis of random arrays. In general, the beampattern of a randomly deployed sensor array will be affected by position errors, amplitude and phase errors, quantization errors and failures of the nodes, which cause a change in the array topology. Throughout this work, the main emphasis will be given to the effect of position errors and solutions to this problem.

1. Position and Phase Errors

There are several references [19], [20], [21], [22] in the literature that deal with position errors in random arrays. The random antenna elements' misplacement causes phase errors and mismatches, which yield degraded performance for the array. In [19], an analysis of the radiation pattern of a random array with both amplitude and phase errors is presented. For an $M \times N$ two-dimensional array, assuming that there are no amplitude errors, i.e., the weights w_{mn} have the same magnitude $I = I_{mn}$, the expected increase Δ_s in the sidelobe level with respect to the main lobe is given by [19]

$$\Delta_s = \frac{1}{MN}\left(e^{\sigma^2_{\Delta\varphi}} - 1\right) \tag{14}$$

where the phase errors follow a Gaussian distribution with zero mean and variance $\sigma^2_{\Delta\varphi}$. For example, in a 5×5 planar array, Δ_s is equal to 0.00518 for $\sigma_{\Delta\varphi} = 20°$ and equal to 0.025 for $\sigma_{\Delta\varphi} = 40°$. Therefore, doubling of the phase error will cause an increase of almost 6 dB in the sidelobe level with respect to the main lobe. However, it can be seen from (14) that the effect of phase errors can be mitigated by increasing the number of array elements in the array. Indeed, Δ_s can be decreased by a factor of two if the number of antenna elements is doubled.

The fractional loss in the main lobe gain due to phase errors is given by [19]

$$\frac{G}{G_0} = e^{-\sigma^2_{\Delta\varphi}} \tag{15}$$

where G and G_o are the main lobe power gains with and without the presence of phase errors, respectively. Since for a 3 dB reduction in the main lobe level, a phase error

standard deviation of $\sigma_{\Delta\phi} = 47°$ is needed, it can be concluded that the main lobe is not significantly affected by random position errors.

2. Random Array Implementation

In general, the effects of misplacement errors depend strongly on the assumptions about the random characteristics of the position deviations from the uniform array. In the previous section, the results were derived based on the assumption of Gaussian phase errors. However, there are several references in the literature which consider different deployments of sensor arrays and consequently different assumptions about the random distribution of the phase errors. One such example is included in [22] where the location of each node in the sensor array is chosen randomly by a uniform distribution within a disk.

Throughout this work, the position errors will be modeled as a deviation from the ideal position of a uniform array. The position errors will be modeled as a uniformly distributed random variable with a minimum value of 0 and a maximum value of $a \times \lambda/2$ where a is the maximum percentage error; therefore, the mean error will be $a \times \lambda/4$. It is also assumed that there is displacement in both x and y directions as indicated in Figure 7. The array axis for randomly placed elements is the best line fit of the sensor nodes' positions and it is assumed to be also the x-axis of the coordinates system. In Figure 7 the best line fit of the nodes location is found and it is assumed to be the array axis. Then the deviations are defined using this array axis as a reference. The beampattern is computed in a plane which is perpendicular to the x-y plane and at ϕ_0 (in this case $\phi_0 = 45°$) with respect to the x-axis.

In Figure 8, the effect of position errors in a 10×1 linear antenna array topology is depicted. The sidelobe levels have been increased and consequently the array performance has been deteriorated. The mean beampattern of a 10×1 linear array with mean position deviation equal to $0.3\lambda/2$ in both x and y directions is compared to the beampattern of the ideal uniform 10×1 array for polar angle $\theta_0 = 30°$ and azimuth angle $\phi_0 = 45°$. The array axis is in the x-direction and the beampattern is in a plane

perpendicular to the x-y plane and at 45° with respect to x-axis ($\phi_0 = 45°$). For the random case, the mean beampattern is obtained after averaging the beampatterns for 50 repetitions of randomly generated array topologies.

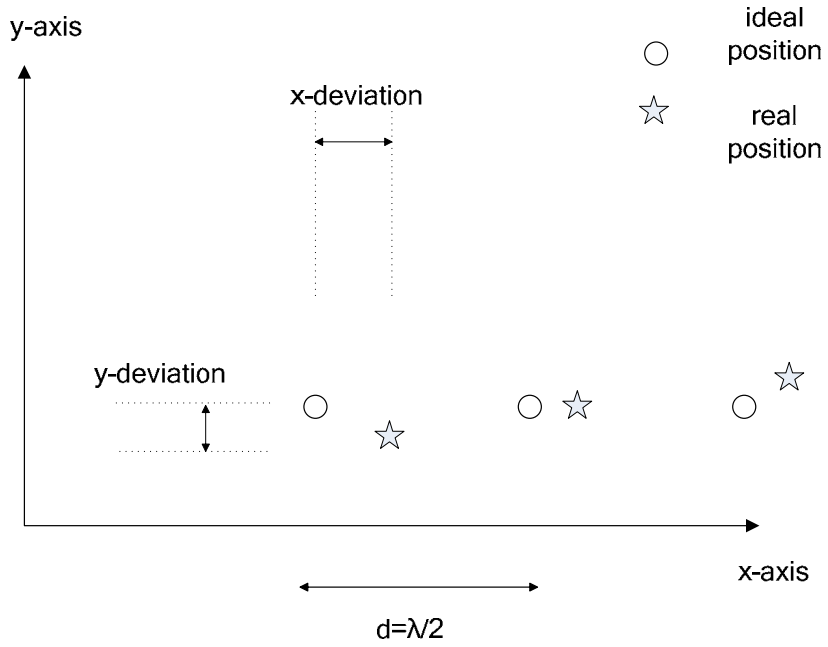


Figure 7. Ideal and approximately linear arrays with randomly deployed elements and position errors.

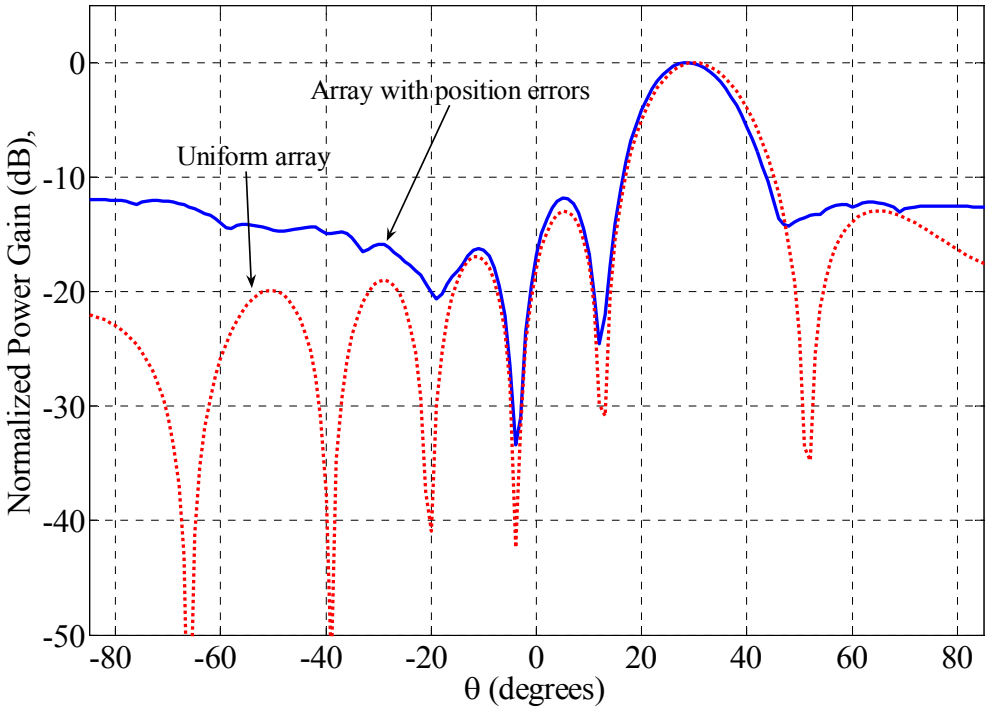


Figure 8. Beampattern of a 10×1 linear array with position errors, compared to a 10×1 array with equally spaced isotropic elements. Polar angle $\theta_0 = 30°$, azimuth angle $\phi_0 = 45°$ and mean deviation 30% of $\lambda/2$ in both x-y directions.

Next, the relationship between the position errors and the increase in the average sidelobe levels is plotted in Figure 9. The deviation from the ideal position is modeled as a uniform random variable in the range of 0 to $\lambda/2$ in both x and y directions, i.e., mean deviation is $0.5\lambda/2$. For each value of mean position error, the increase for the first sidelobe (red line) and the largest of all the sidelobes (blue line) is calculated by averaging 50 simulation runs. It is obvious that the sidelobe power gain increased significantly as the mean deviation increased; for example, if the mean deviation is about $0.4\lambda/2$, then the maximum sidelobe increase is almost 5.3 dB while the first sidelobe increase is about 4.2 dB. Thus, the strongest sidelobe is lower from the main lobe by 7.7 dB only while the first sidelobe differs from the main lobe by 8.8 dB.

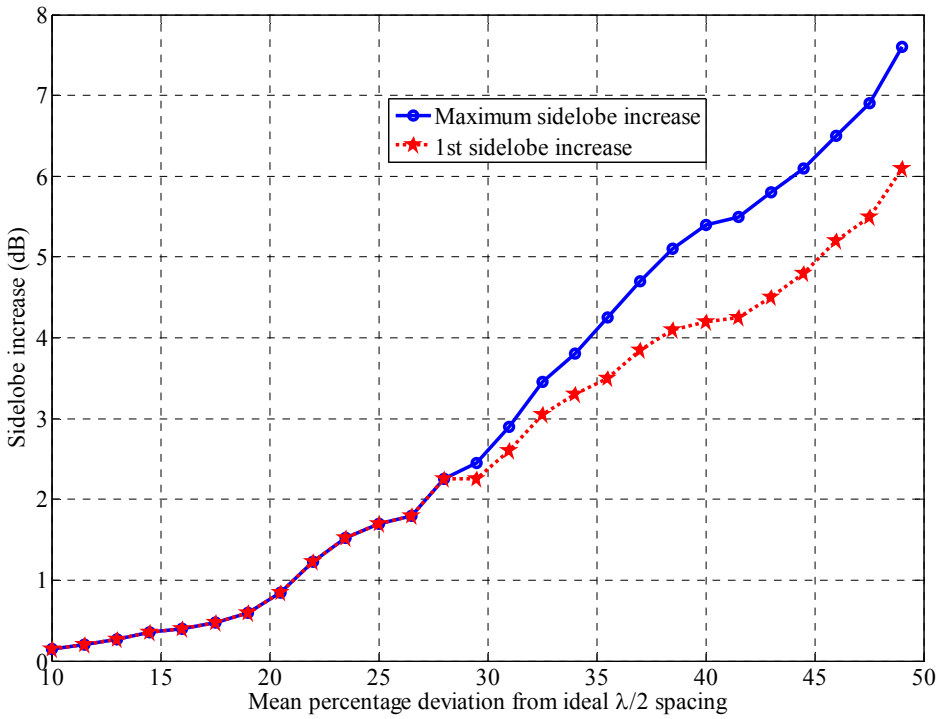


Figure 9. Effect of position errors in a random array. Average sidelobe level (dB) for the first (red line) and the largest (blue line) sidelobe as a function of the mean deviation of the actual element positions from the ideal positions in linear 10×1 array with equally spaced isotropic elements.

C. BEAMFORMER IMPLEMENTATIONS

In the previous sections, it was shown that the beampattern of an antenna array is determined by the direction of the incoming signal, the number of the array elements $M \times N$, and the array topology, which includes the position errors and the set of weights w_{mn}. The objective of a beamformer is to preferentially receive a signal from a specific direction or to preferentially transmit a signal in that direction. It is also usually desirable to suppress interference signals, which come from other directions. Therefore, the beamforming operation consists of adjusting the weights w_{mn} in such way that the main lobe is steered towards the desired signal's AOA. Such a beamformer for M elements in a linear array is depicted in Figure 10, and it is typically used for narrowband signals.

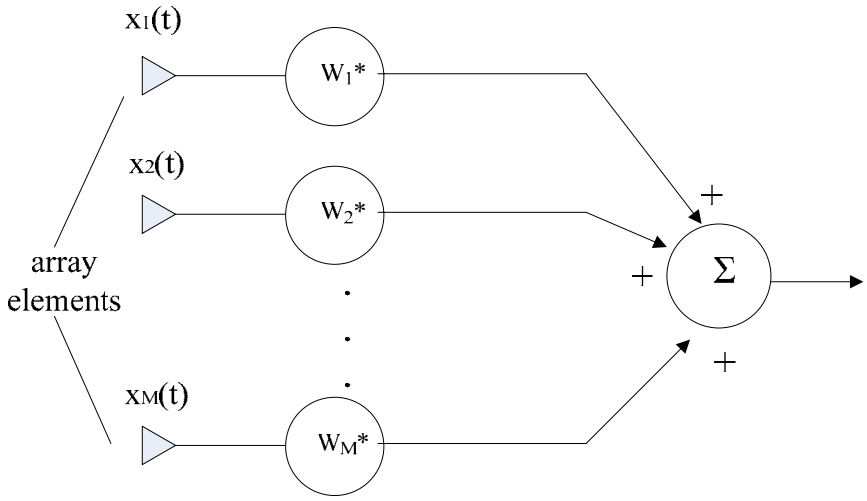


Figure 10. A $M \times 1$ narrow-band beamformer.

The output of the beamformer is given by

$$y(t) = \underline{w}^H \underline{x}(t) \tag{16}$$

where $\underline{w}$ is the weight vector and $\underline{x}(t)$ is the signal vector

$$\underline{x}(t) = [x_1(t) \quad x_2(t) \quad \ldots \quad x_M(t)]^T . \tag{17}$$

A conventional beamformer can be described [8] as a delay-and-sum beamformer with all weights having the same magnitude. The phases are selected to steer the array beampattern towards a desired direction (θ_0, ϕ_0). However, there are many types of beamformers, which can be classified as either data independent or statistically optimum, depending on how the weights are chosen [11].

In a data independent beamformer, the weights are chosen so as to create a specified desired response for all signal and interference cases. The array data (the signal vector $\underline{x}(t)$) are either not known or not taken into account for the beamforming design. If the desired response is $F_d(\theta, \phi)$ in order to receive a signal from a certain direction and cancel out interferences from other directions, then the weight vector $\underline{w}$ is chosen in such a way that the actual response $F(\theta, \phi)$ approximates the desired one. In the following

chapters, the beamforming operation will be based on data independent techniques, which try to create an approximation of the desired response $F_d(\theta,\phi)$.

The second class of beamformers [11] contains the statistically optimum ones. In this case, the weights are chosen based on the statistics of the array data, and the goal is to optimize the array response using several criteria. Generally, a statistically optimum beamformer tries to cancel out the interfering signals by placing nulls in their incoming directions in order to maximize the Signal to Noise Ratio (SNR) at the output of the beamformer. Such a beamformer is the Multiple Sidelobe Canceller (MSC), which needs auxiliary channels free of the desired signal but is very simple in its implementation. Another optimum beamformer requires knowledge of both the desired signal and noise covariance matrices R_s and R_n, respectively, and has the advantage that it maximizes the SNR, but it is not easy to implement. The Minimum Variance Distortionless Response (MVDR) beamformer computes the weights given a set of constraints and provides a very good performance, but it is also computationally intensive.

There are also adaptive algorithms for beamforming, which compensate for the fact that the signal's statistics are usually not known or may vary with time. Such beamformers for the weight determination are based on the well-defined and popular LMS and RLS algorithms and also on numerous variations of them. A more thorough analysis of these adaptive algorithms and of the previously described statistically optimum beamformers can be found in, [11], [12] and [14] .

D. WIRELESS SENSOR NETWORK ARRAYS

This section presents a discussion of sensor node deployment schemes and the concepts of the processing and communication costs which will be used for the performance evaluation of the beamforming algorithms.

1. Deployment of Sensor Nodes

As mentioned in Chapter I, recent developments in the MEMS technology have enabled the construction of small, cheap, multifunctional sensors with signal processing

and communication capabilities. These sensor nodes can be deployed over an area of interest and can be used in order to collect process and transmit information. However, there are still many challenges for the efficient implementation of a sensor network, such as efficient power consumption, controllable deployment of the nodes, source localization, self organization, and others.

In this work, the specific application scenario which is examined is summarized in Figure 11. A set of sensor nodes are deployed in the battlefield, and they acquire the desired information, which may be any type of signal (acoustic, video, etc.). The UAV, flying over the sensor field, establishes a connection with the network in order to obtain the collected data. Due to the limited sensor capabilities, a single node can not transmit its data directly to the UAV since it does not have the required transmission range. However, the nodes can be organized into a large antenna array where each sensor plays the role of an antenna element and implements a distributed beamformer.

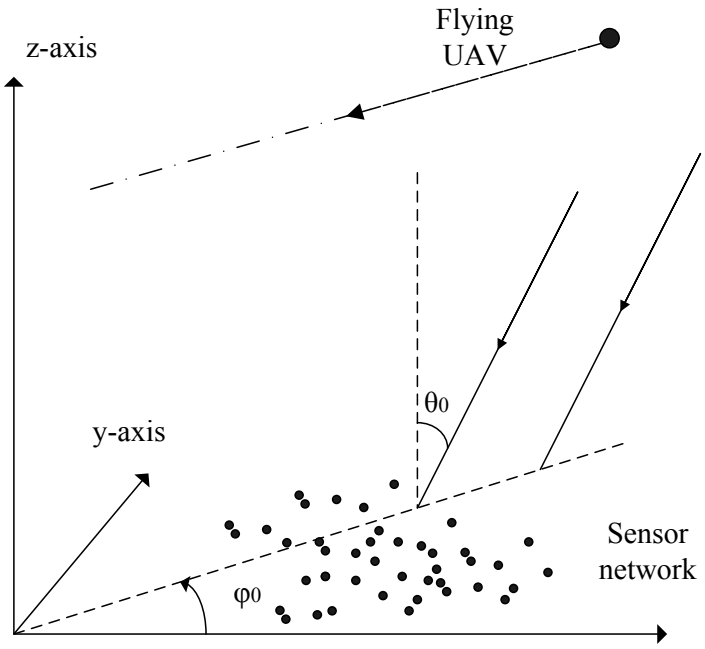


Figure 11. A sensor network deployment used for information transfer to a UAV (After Ref. [34]).

In the previous section, it was shown that the best performance for an antenna array is achieved when the elements are located on a rectangular grid with an interelement distance equal to $\lambda/2$. Nevertheless, it is obvious that this ideal deployment can rarely be achieved in a real operational environment. The nodes may be dropped by the UAV or deployed by a ground force and consequently the sensor array topology cannot be the desired one. This randomness imposes position errors, which effect the beampattern of the sensor array as analyzed in the previous section.

Much work has been done in analyzing the dependence of the random array response on the statistical characteristics of the array topology [19], [20], [21],[22]. Furthermore, much emphasis has been placed on algorithms that allow a node to define its position with respect to a reference node within the sensor network [23]. In the next chapter, several beamforming algorithms that use location information will be described. Assuming that the coordinates of a node in a local coordinate system are known, there are many schemes for successfully mitigating the effect of the position errors and choosing a suitable set of weights for the beamformer. However, there are limitations in the performance of these algorithms when the position errors are large enough.

Another practical approach could be to search and locate a suitable subset of nodes and then form an antenna array. In [24] and [25], a central node (cluster head) tries to find a set of nodes whose topology is close to a uniform linear or planar array according to some geometric criteria. Using only the distance between the nodes as known information, the proposed algorithm tries to find the optimum subset of sensors with minimum mean position error. Figure 12 illustrates this concept, where a set of five sensors yielding the best approximation to a 5×1 uniform linear array is determined. After finding this optimum set of nodes, several beamforming schemes can be applied in order to find the weight coefficients. Therefore, the combination of finding a suitable set of nodes with an appropriate beamforming technique can provide sufficient performance.

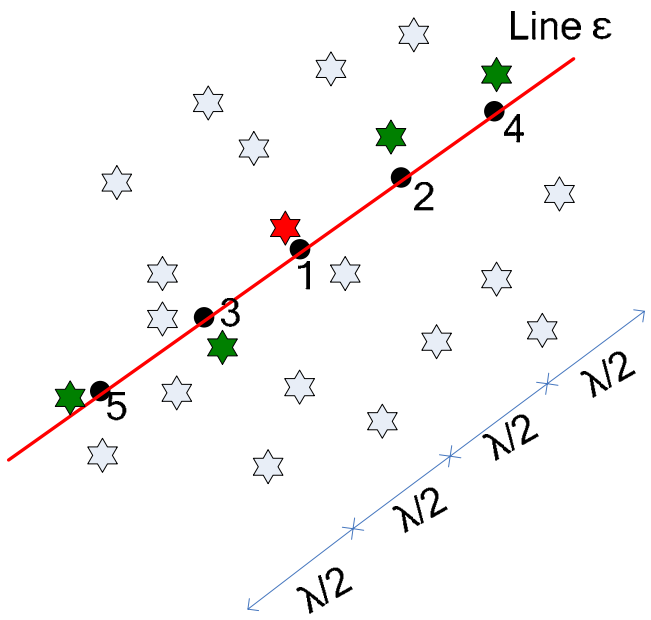


Figure 12. Finding five nodes approximating a 5×1 linear array with equally spaced elements (After Ref. [24]).

2. Communication and Computational Cost

An important issue in the sensor networks is the efficient energy consumption by the nodes. Low-power hardware components and low-duty cycle operation techniques must be applied in order to achieve the required ultra-low-power operation. Energy is consumed while processing or transmitting data. Therefore, the beamforming algorithms must be evaluated in terms of realistic power consumption. In order to achieve this goal, it is obvious that there is a need for schemes implemented in a distributed manner; thus, the computational effort and consequently the energy needed is shared among the sensors. Moreover, these techniques should minimize the communication since wireless transmission consumes considerable amount of power during a node's operation.

The need for distributed algorithms in order to minimize the communication power consumption is discussed in [26]. A radio transmitting 1 kb of data over a distance of 100 m, with an operating frequency of 1 GHz using BPSK modulation having an error probability of 10^{-6} and fourth-power distance loss with Rayleigh fading, requires

approximately 3 Joules of energy. The same amount of energy can perform 300 million instructions for a 100 MIPS/watt general processor [26]. This results in a ratio of 30,000 processing instructions per transmitted bit with equal energy consumption. Other practical implementations [26], [27] have yielded ratios from 200 to 3000. This ratio of communication cost to computational cost depends largely on the sensor characteristics (transmission range, complexity of the instruction in number of bits, etc.) and can be increased in the presence of noise where retransmissions will be needed.

In general, the relationship of the communication and computational cost with the power consumption depends on the technical specifications of the sensors, the applications, and other factors that are difficult to exactly predict, such as the presence of noise. However, a general framework can be formed using the definitions in Table 1.

Symbol	Definition
P_{tb}	Mean power per transmitted bit (power consumed to transmit a number of bits divided by this number)
N_{tb}	Number of transmitted bits
P_i	Mean power per instruction in the sensor's processor (power consumed to perform a number of instructions, divided by this number)
N_i	Number of instructions
P_c	Total transmission power or communication load
P_p	Total processing power or computational load
P	Total power for a specific application

Table 1. Quantities used for defining the communication and computational cost.

The transmission power P_c is given by

$$P_c = N_{tb} \times P_{tb} \tag{18}$$

and the processing power is given by

$$P_p = N_i \times P_i \tag{19}$$

while the total power is the sum of these two factors

$$P = P_c + P_p. \tag{20}$$

The parameter η_{tp} is defined as the ratio of the power per transmitted bit to the power per instruction, and as mentioned before, it can vary from 200 to 3000, indicating that one transmitted bit requires much more energy than one performed instruction by the sensor's processor.

$$\eta_{tp} = \frac{P_{tb}}{P_i} \ . \tag{21}$$

Throughout this work the communication cost is focused on the implementation of the algorithms. Therefore, it depends only on the data elements that need to be transmitted for the implementation of the different beamforming schemes. These data elements must be encapsulated in data packets, thus there is a communication overhead due to these packets. Furthermore, the organization of the sensor nodes in a cluster needs also a number of control packets. This networking cost, which depends on various factors as the number of sensors, the noise and the protocols used, will not be taken into account throughout this work and will be left for future analysis.

Using the above definitions, the total power consumption for the implementation can be calculated and the different beamforming methods can be compared using a common framework.

E. SUMMARY

In this chapter, the basics of beamforming in antenna arrays were discussed; specifically, the beampatterns of the uniform linear and planar arrays were presented. This was followed by an analysis of the effect of the position errors in random arrays on the array performance as measured by an increase in the sidelobe levels. Beamformer implementations were discussed, including a brief reference to various techniques that have been reported in the literature. The specific operational scenario for communication between sensor networks and UAV was also briefly presented. Finally, the communication and computational cost as functions of the power consumption were introduced as metrics for the evaluation of beamforming algorithms in sensor networks. The strict requirements for low power consumption by the sensor nodes create the need for distributed beamformimg algorithms (presented in Chapter IV) compared to centralized algorithms (presented in Chapter III).

III. CENTRALIZED IMPLEMENTATION OF A BEAMFORMER

This chapter is focused on centralized implementation of the beamforming operation for linear arrays. In the centralized approach, it is assumed that all information needed to determine the weight vector is available in a specific node, which can be the cluster head in a sensor network. This node will collect all the data, such as the steering vector $\underline{d}(\theta,\phi)$, which depends on the array topology, the direction of the desired signal (θ_0,ϕ_0) and the direction of potential interferences, and will calculate the weight vector to provide the desired array response at the output of the beamformer.

Two different implementations are presented and their advantages and disadvantages, such as simplicity in implementation, overall performance and computational demands, are discussed. Finally, an analysis of the computational and communication cost examines the feasibility of these implementations in sensor networks where power efficiency is a major issue.

A. PHASE MATCH OF THE STEERING VECTOR

This is the simplest implementation of a beamformer and can be considered an extension of the conventional beamformer for the case of random position errors. In this method, as in the case of a uniform array (see Chapter II), the weights of the elements $\underline{w}$ are chosen with the goal to match the steering vector $\underline{d}(\theta_0)$ and create the main lobe with maximum gain towards the AOA θ_0. If the steering vector is

$$\underline{d}(\theta_0) = [1 \quad e^{j\beta x_2 \sin\theta_0} \quad e^{j\beta x_3 \sin\theta_0} \quad ... \quad e^{j\beta x_n \sin\theta_0}]^T \qquad (22)$$

then the weights are $\underline{w} = \underline{d}(\theta_0)$. Each element of the weight vector has unit magnitude and the same phase as the corresponding element of the steering vector. Thus, the array response

$$F(\theta) = \underline{w}^H \underline{d}(\theta) \qquad (23)$$

has its maximum $F(\theta_0)$ at the AOA θ_0.

Due to position errors in the sensor array, the distances $x_1, x_2 ..., x_n$ from the reference node are not multiples of the ideal distance $\lambda/2$. However, the weight vector can still be selected to match the steering vector in the desired direction of θ_0. The implementation of this approach in a sensor network is very simple and assumes the following:

a) Each node can calculate its position from a reference node accurately.

b) The relative positions of the nodes are disseminated to a central processing node, a cluster head if a cluster hierarchical architecture is established.

c) The desired steering direction of the incoming signal is known to the central node.

d) Errors due to noise during the communication among the nodes are not taken into account.

The cluster head (central node) collects all the necessary information and calculates the weight vector. Then it sends to each node its corresponding weight, which will be used for the beamforming and the steering of the sensor array.

The array pattern of this simple beamformer is shown in Figure 13 for a 10×1 linear array with uniform position errors up to $0.4\lambda/2$. The signal's direction of arrival is known ($\theta_0 = 30°$, $\phi_0 = 45°$). The figure shows the mean beampattern averaged over 50 simulation runs.

Some conclusions can be drawn for this beamforming technique. First, it is simple in implementation since the central processing node needs only to receive the relative position from each node one at a time and then send back the calculated weights to each node. Second, the main lobe of the beampattern has the maximum value in the direction of arrival (θ_0, ϕ_0), and it is exactly equal to the maximum value of the (ideal) uniform array. However, the beampattern presents notable deviations for angles other than the AOA. In this specific case, the sidelobe power level increases significantly for angles lower than $0°$. In general, the performance of the array approaches that of the uniform array at angles around the AOA but deteriorates for other elevation angles.

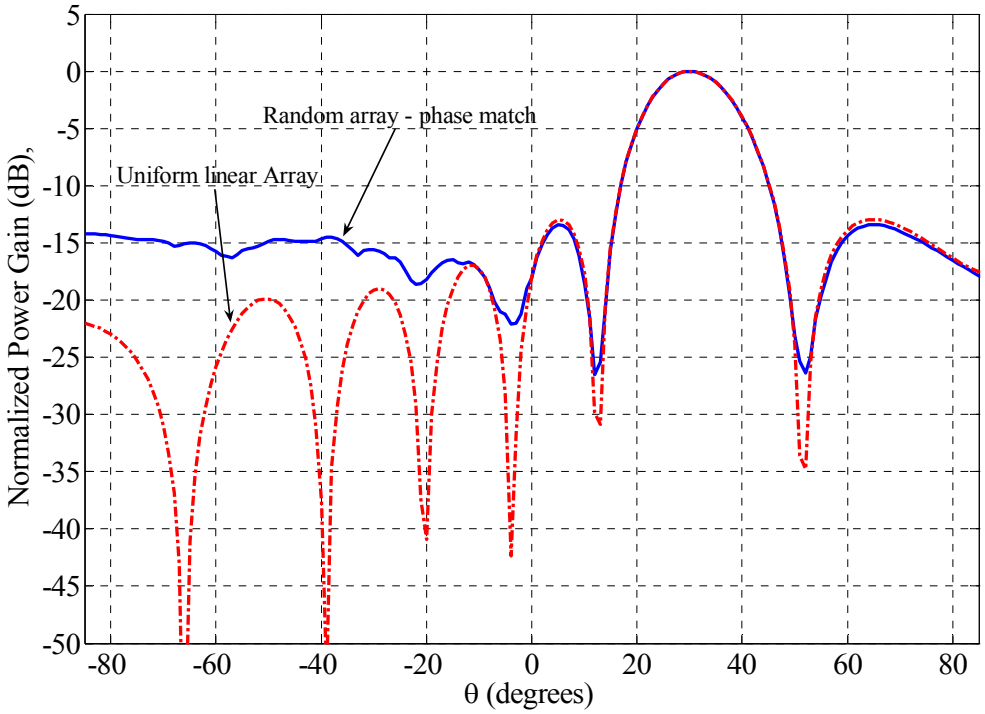


Figure 13. Mean beampattern of a random 10×1 uniform linear array with position errors, averaged over 50 simulation runs, compared to a uniform 10×1 array of equally spaced elements. The AOA is ($\theta_0 = 30°$, $\phi_0 = 45°$). The position errors are uniformly distributed between 0 and and $0.4\lambda/2$ in both x and y directions. Beamforming is implemented by matching the steering vector.

B. BEAMPATTERN APPROXIMATION IN THE LEAST SQUARE SENSE

The beamformer presented in the previous section computes the weights of the antenna elements by trying to achieve the best performance in the direction of the signal arrival and does not set any requirement for the other directions. Although it behaves well at angles around the desired direction, it suffers from high sidelobes in other directions. Therefore, it cannot be used when there are interference signals coming from directions other than the signal's AOA or if the desired signal comes from a certain range of directions. If there is a strong source of interference, the desired array response should be zero in that direction, in order to cancel the interfering signal. As a result, there is a need for a beamforming design, which calculates the weight coefficients $\underline{w}$ in such way that the resulting response approximates the desired response over a range of directions.

1. Least Squares Problem Formulation

As mentioned before, the array response for a specific angle θ is given by

$$F(\theta) = \underline{w}^H \underline{d}(\theta) . \tag{24}$$

If the desired response $\underline{F}_d(\theta)$ is defined over m number of angles, then the actual array response $\underline{F}(\theta)$ of n antenna elements for these m angles is

$$\underline{F}(\theta)^H = \underline{w}^H D(\theta) \tag{25}$$

where

$$\underline{F}(\theta) = \left[F(\theta_1) \quad F(\theta_2) \quad \cdots \quad F(\theta_m) \right]^T \tag{26}$$

is a $m \times 1$ vector which contains the array response for each one of the angles θ_i, with $i = 1, \ldots m$. The weight vector $\underline{w}$ is an $n \times 1$ column vector and $D(\theta)$ is an $n \times m$ steering matrix containing the steering vectors $\underline{d}(\theta_i)$ for each one of the angles θ_i:

$$D(\theta) = \left[\underline{d}(\theta_1) \quad \underline{d}(\theta_2) \quad \cdots \quad \underline{d}(\theta_m) \right] \tag{27}$$

or

$$D(\theta) = \begin{bmatrix} e^{j\beta x_1 \sin\theta_1} & e^{j\beta x_1 \sin\theta_2} & \cdots & e^{j\beta x_1 \sin\theta_m} \\ e^{j\beta x_2 \sin\theta_1} & e^{j\beta x_2 \sin\theta_2} & \cdots & e^{j\beta x_2 \sin\theta_m} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\beta x_n \sin\theta_1} & e^{j\beta x_n \sin\theta_2} & \cdots & e^{j\beta x_n \sin\theta_m} \end{bmatrix} . \tag{28}$$

Note that for simplicity the array response is defined over a set of arrival angles $\theta_1, \ldots, \theta_m$ while the azimuth angle ϕ_0 is fixed. If the desired response is defined over a combination of m elevation angles $\theta_1, \ldots, \theta_m$ and q azimuth angles $\phi_1, \ldots, \phi_q$, then the array response $F(\theta, \phi)$ is a $m \times q$ matrix and the steering matrix $D(\theta, \phi)$ is an $n \times m \times q$ 3-D matrix. This adds unnecessary complexity to the formulation of the problem and is to be avoided. In the following analysis, the array response will be defined for a known, fixed azimuth angle ϕ_0 and a set of m arrival angles $\theta_1, \ldots, \theta_m$.

If the desired response is $\underline{F}_d(\theta)$, the goal is to select the weights $\underline{w}$ such that the actual response $\underline{F}(\theta)$ generated by this set of weights approximates the desired response. This can be done by trying to minimize the L_2-norm of the difference between the desired and the actual response. In other words, the weight coefficients $\underline{w}$ are chosen in order to minimize the mean squared error between $\underline{F}_d(\theta)$ and $\underline{F}(\theta)$

$$\varepsilon = \min_{\underline{w}} \left| \underline{F}(\theta) - \underline{F}_d(\theta) \right|^2 . \tag{29}$$

Therefore the problem of the antenna weight calculation is solved using a classic least squares (LS) procedure. Taking the transpose of (25), the array response can be written as

$$\underline{F}(\theta) = D(\theta)^H \underline{w} . \tag{30}$$

For m number of angles or approximation points and n number of sensors, with $m > n$, the problem becomes an overdetermined LS problem

$$\varepsilon = \min_{\underline{w}} \left| D(\theta)^H \underline{w} - \underline{F}_d(\theta) \right|^2 \tag{31}$$

where again $\underline{F}(\theta)$ is a $m \times 1$ vector and $D(\theta)^H$ is a $m \times n$ matrix. Provided that the $n \times n$ matrix $D(\theta)D(\theta)^H$ is invertible (i.e., $D(\theta)^H$ is full rank), then the solution to the LS problem of (30) is given by

$$\underline{w} = D^+(\theta)\underline{F}_d(\theta) \tag{32}$$

where $D^+(\theta)$ is the pseudo inverse of $D(\theta)^H$ defined as

$$D^+(\theta) = (D(\theta)D(\theta)^H)^{-1} D(\theta) . \tag{33}$$

The weight coefficients that have been calculated using the LS approach have both different amplitudes and phases. In the array with equally spaced elements, the amplitude I_m was the same for all elements ($I_m = 1, \forall m$) and only the phase was different. However, in the LS approach the weights have also different amplitudes. So in the LS approaches the sensors modify both amplitudes and phases in order to approximate the desired response.

The design of an array using the above decribed LS approximation of the desired array response is related to the design of arrays using the Fourier analysis. In this approach, the desired array function, which is required to create the desired array response, can be expanded in a Fourier series. Truncating this Fourier series, the result is an array with a finite number of elements. Such an array is optimum in the sense that no other array with the same number of elements can approximate the desired array function with lower mean squared error.

2. Implementation and Performance Analysis

The significant advantage of the above LS approach is that the desired response can be closely approximated over a set of angles, thus providing the ability to cancel out unwanted signals and amplifying only the desired one. Results of such an implementation of the LS scheme for a 10×1 random array can be seen in Figure 14, where the desired response $\underline{F}_d(\theta)$ is the array response of the (ideal) uniform 10×1 linear array. For the approximation of the desired beampattern 180 points (angles from $-90°$ to $90°$) were used. The mean beampattern is averaged over 50 simulation runs, and there is virtually no difference between the desired (red line) and the actual beampattern (black line) as generated by the weights computed by minimizing the LS criterion. The actual response can be also compared with the response derived by the technique of the phase match approach from the previous section (cyan line).

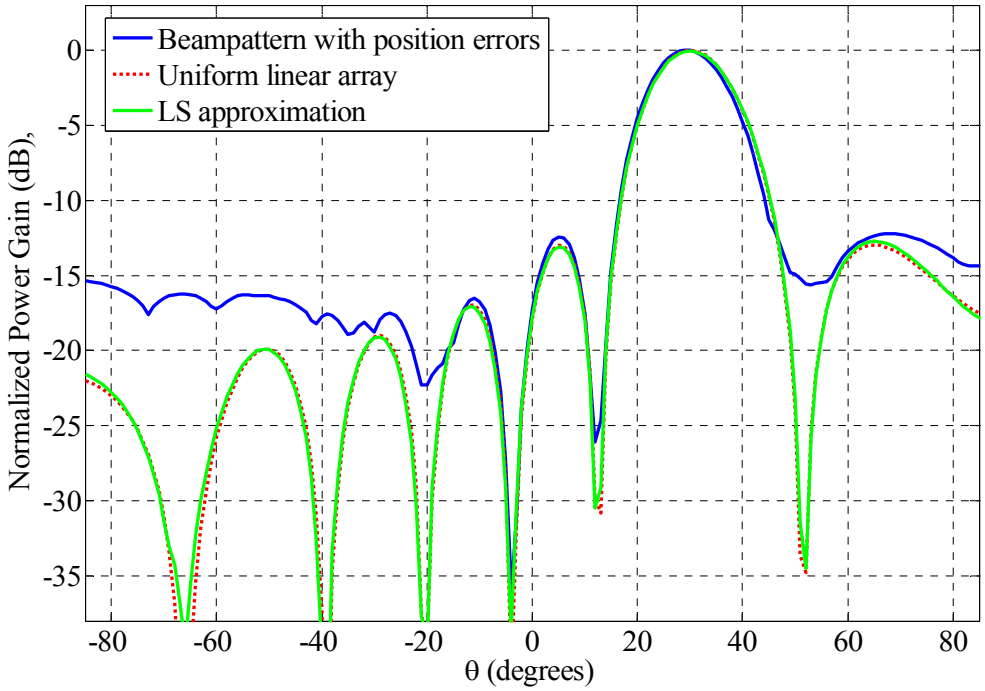


Figure 14. Mean beampattern of a random 10×1 linear array with position errors (blue) and mean beampattern calculated using the "phase match" (cyan) and the LS approach (red), averaged over 50 simulation runs, compared to a uniform 10×1 array (black). The AOA is ($\theta_0 = 30°$, $\phi_0 = 45°$). The position errors are uniformly distributed between 0 and and $0.5\lambda/2$ in both x and y directions

The performance is almost the same for both phase match and LS schemes around the desired AOA, but the LS approach is obviously better in any other direction. The phase match technique is more vulnerable to interference signals, especially to those coming from $\theta < 0°$, while the LS approach creates a beampattern almost identical to the desired response of a uniform array. Indeed, the determination of the weights using the LS approximation technique achieves elimination of the effect of the position errors (blue line); the position errors are uniformly distributed between 0 and $0.5\lambda/2$ in both x and y directions.

The LS approach is clearly a centralized approach. The implementation in a sensor network makes similar assumptions as in the phase match approach including an additional assumption: the desired array response defined over a set of angles $\theta_1,\ldots,\theta_m$ is

assumed to be known to the cluster head. This response may include the direction (θ_0, ϕ_0) of the incoming signal and/or the directions of interfering sources.

The cluster head collects all the necessary information and specifically the relative positions $x_1, \ldots, x_n$ of the nodes and the desired response $\underline{F}_d(\theta)$. The steering matrix $D(\theta)$ is completely defined by the positions and the desired angles for the beampattern approximation. After collecting all necessary data, the cluster head constructs the steering matrix using the position data and the set of angles and calculates the weight vector by solving the LS problem of (30). Each node receives the corresponding coefficient from the cluster head, and the sensor array is ready to transmit or receive towards the desired signal's direction and simultaneously suppress potential interfering sources.

The LS approach's performance is satisfactory in eliminating the effect of the position errors as observed in Figure 14. The assumptions mentioned before, such as that information about the relative position of the nodes and the desired response (incoming direction of signal and interferences), are reasonable in a sensor network environment. There are algorithms which allow a sensor node to calculate its relative position from a cluster head using localization techniques [23], [26]. Thus, the LS scheme could be implemented in a cluster-based sensor array where all information is collected and processed in the cluster head.

Nevertheless, this centralized approach suffers from the inherent problems of any scheme where the processing effort is not distributed among the nodes but performed by a single node (cluster head). Since the main effort is undertaken by one sensor only, a single point of failure is created in the sensor array. If this specific node fails, then the LS problem has to be solved from the beginning, i.e. a new central sensor must collect all the information, calculate the weight coefficients and disseminate them to the other nodes. This is a waste of valuable processing and communication resources which are very restricted in the sensor nodes. Considering that failure of sensor elements in a sensor array is something common due to their low battery life and low-cost construction, this centralized approach lacks robustness.

Furthermore, since new nodes may offer to participate in the sensor array or other nodes may decide to switch into sleep mode due to limited power, the array topology will change frequently and dramatically. This means that, for any modification in the sensor array, the LS problem must be solved from the start in order to determine the new set of weight coefficients. For this reason, the centralized approach does not offer the required scalability, which is desired in a sensor network environment.

Finally, the processing load of solving the LS problem increases exponentially as the number of sensors n and the number of approximation points (angles) m increase. If this processing effort is performed by a single sensor, then its power will be consumed rapidly and the sensor will fail.

The solution is a distributed implementation of the LS approach, and such two schemes will be presented in Chapter IV. First, the single point of failure limitation can be overcome by distributing the processing load among many nodes. The total processing effort in a distributed approach may be higher compared to the centralized approach, but the tradeoff is the increased robustness of the system. Furthermore, the LS approach must be implemented in such way that changes in the array topology do not require the solution of the problem from scratch but only small modifications to the already calculated weight coefficients. This can be achieved by an algorithm implemented in a distributed fashion, which will offer considerable scalability to the problem. Since the LS approach performs very well in approximating the desired pattern, a distributed implementation of it will first retain this feature and second, it will offer the desired scalability and robustness; thus, it can provide a reliable solution for beamforming in sensor networks.

In the next section, a discussion about the processing cost as a function of the number of approximation points is presented. As mentioned before, the processing load increases dramatically with the number of approximation points; therefore, the minimum number of points should be used in order to reduce the computational effort.

C. APPLICATION OF LEAST SQUARES BEAMFORMING APPROACH TO SENSOR NETWORKS

In the previous section the LS approach was presented for the approximation of a desired response defined over a set of direction angles or approximation points. As mentioned before, the processing load of the LS approach depends largely on the number of nodes and the number of approximation points. For example, if n is the number of sensors and m the number of angles, the solution of the LS problem using the normal equations needs $(m+n/3)n^2$ flops [28]. By using fewer approximation points, the processing load can be reduced, which is crucial for the sensor nodes with limited processing power. However, the solution will not be the same as the unmodified case, and the approximation will degrade as the number of points is decreased. Therefore, it is worth examining the "quality" of the desired response approximation as a function of the number of approximation points.

In order to illustrate this relationship, a series of 50 simulations with corresponding random arrays was performed. In each run, the desired response of a uniform linear 10×1 array (ideal) was approximated using different number of approximation points ranging from 10 to 90. For each set of approximation points, the corresponding solution for the weight coefficients was derived using the LS approach. The reference approximated response and the reference weight vector was based on 180 approximation points (one for each degree from $-90°$ to $90°$). The solution of the LS approach for each set of approximation points is compared to the reference solution derived from 180 points.

Two error metrics are used to evaluate the performance of each set of approximation points. The first metric e_w is the mean L_2-norm of the difference between the reference weights and the examined weights

$$e_w = \frac{1}{S}\sum_{1}^{S}\sqrt{\left|\underline{w}_i - \underline{w}_{180}\right|^2} \tag{34}$$

where $\underline{w}_{180}$ are the weights derived from the 180-point approximation (reference), $\underline{w}_i$ are the weights derived by an i- point approximation, for $i = 10,\ldots,90,$ and $S = 50$ is the number of simulation runs.

The second metric e_F is the mean L_2-norm of the difference between the reference array response derived from 180 approximation points and the array response for fewer approximation points (10 to 90)

$$e_F = \frac{1}{S}\sum_{1}^{S}\sqrt{\left|\underline{F}_i(\theta) - \underline{F}_{180}(\theta)\right|^2} \tag{35}$$

where $\underline{F}_{180}(\theta)$ is the approximated array response using 180 points (reference) and $\underline{F}_i(\theta)$ is the response derived by an i- point approximation, for $i = 10,\ldots,90$.

The test topology is a 10×1 random array where the antenna elements have position errors, modeled as uniform random variables between 0 and $0.4\lambda/2$. The results are shown in Figures 15 and 16 for the two errors e_w and e_F, respectively.

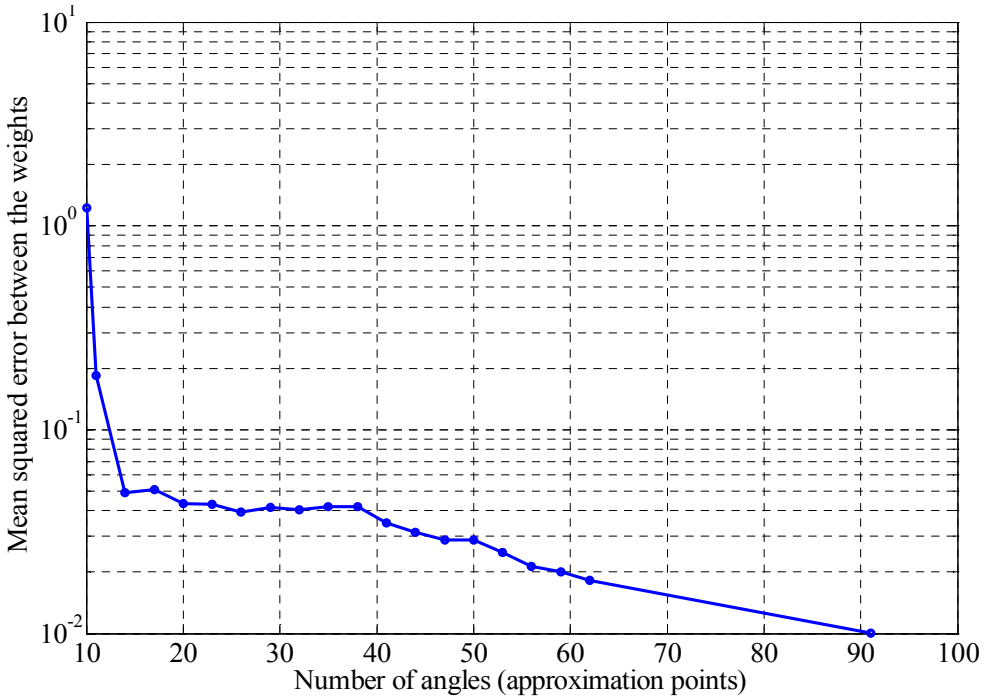


Figure 15. Error metric e_w of weights as a function of the number of approximation points m.

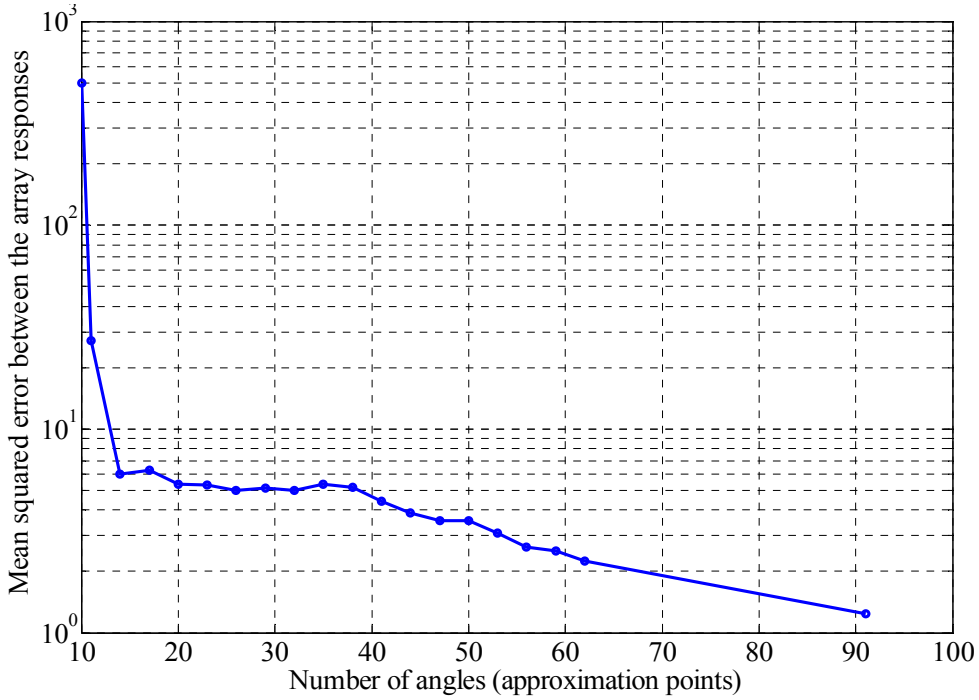


Figure 16. Error metric e_F of approximated array responses as a function of the number of approximation points m.

In Figure 15, error metric e_w decreases as the number of approximation points increases, which is reasonable. The significant result comes from the fact that the difference between the reference weights and the weights computed for fewer approximation points is very small, even for 15 approximation points. This means that using only 15 approximation points, the weight vector is very close to the reference weight vector (180 approximation points). Therefore, there is no need to use 180 points to approximate the desired response since this can be done by using only 15 points, which implies that the processing cost can be significantly reduced.

Similar comments can be made about the results in Figure 16 for the error between the reference array response derived by 180 approximation points and the responses derived by fewer points. If the desired response is known and needs to be approximated using a LS approach for the determination of the weights, the number of approximation points can be substantially lowered without significant degradation in the

performance of the antenna array. Thus, the processing cost can be reduced, which is desirable in a sensor network environment with restrictions on the power consumption.

D. SUMMARY

In this chapter, two centralized approaches for the computation of the weights were presented. The first technique is simple and easy to implement, but the performance is not satisfactory in the presence of interference signals. The second approach is based on the LS approximation of a known desired response by selecting the weights in order to minimize the error between the actual and the desired response. Although having satisfactory performance, the LS scheme still lacks robustness and scalability since it is a centralized approach. Finally, the performance of the approximation of the desired response as a function of the number of approximation points was examined. The next chapter presents two algorithms that implement the LS approach in a distributed fashion.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DISTRIBUTED ALGORITHMS FOR BEAMFORMING

A Least-Squares (LS) based centralized approach for solving the beamforming problem was described in the previous chapter. Nevertheless, in a sensor network environment where the processing and communication load must be shared among the nodes, there is a need for distributed beamforming algorithms.

Two distributed implementations of the LS beamforming problem are proposed in this chapter. The first is based on the well known QR factorization using Householder transformations, where each node performs a part of the QR decomposition process. The second approach is a distributed iterative solution of the LS problem, which converges quickly to the actual weight coefficients. Both algorithms efficiently distribute the processing load among the nodes; however, the tradeoff consists of an increase in the communication cost.

A. DISTRIBUTED QR FACTORIZATION WITH HOUSEHOLDER TRANSFORMATIONS

1. Householder Transformations

A Householder transformation (or Householder reflection) is a transformation of reflecting a vector about some known plane [28], [29]. Given an arbitrary vector $\underline{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}^T \in \mathbb{R}^m$ and a unit vector $\underline{e}_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T \in \mathbb{R}^m$, the $m \times m$ Householder matrix is defined as the matrix that transforms $\underline{x}$ to a vector parallel to $\underline{e}_1$

$$H\underline{x} = \alpha \underline{e}_1 \tag{36}$$

where α is a scalar.

A Householder matrix can be formed by [29]

$$H = I - \frac{1}{\beta}\underline{v}\underline{v}^T \tag{37}$$

where β is a scalar, I is an identity matrix and $\underline{v}$ is an $m \times 1$ vector. From the above definition, the Householder matrix H is completely determined by vector $\underline{v}$ and scalar β, so there is no need to store all m^2 elements of H, but only $\underline{v}$ and β, which are only $m+1$ elements.

The significant characteristic of the Householder matrix is that it is an orthogonal matrix; thus, it has the property

$$H^T = H^{-1} = H \quad . \tag{38}$$

Orthogonal matrices or orthogonal transformations can be used to obtain a QR factorization of a matrix A, and this in turn can be used to solve a linear system $A\underline{x} = \underline{b}$, as described in the following section.

The above defined Householder matrix zeros out all the elements of an $m \times 1$ vector $\underline{x}$ except the first one, but one can construct Householder matrices that zero out only the last $m-k$ components of a vector $\underline{x}$ [29]. Let $\underline{x}^{(1)}$ and $\underline{x}^{(2)}$ define

$$\underline{x}^{(1)} = \begin{bmatrix} x_1 & x_2 & \cdots & x_{k-1} \end{bmatrix}^T \tag{39}$$

$$\underline{x}^{(2)} = \begin{bmatrix} x_k & x_{k+1} & \cdots & x_m \end{bmatrix}^T \tag{40}$$

while $I^{(1)}$ and $I^{(2)}$ denote $(k-1)\times(k-1)$ and $(m-k+1)\times(m-k+1)$ identity matrices, respectively. From (32), an $(m-k+1)\times(m-k+1)$ Householder matrix $H_k^{(2)}$ can be constructed as [29]

$$H_k^{(2)} = I^{(2)} - \frac{1}{\beta_k} \underline{v}_k \underline{v}_k{}^T \tag{41}$$

such that

$$H_k^{(2)} \underline{x}^{(2)} = \left\| \underline{x}^{(2)} \right\|_2 \underline{e}_1 \ . \tag{42}$$

By defining the $m \times m$ matrix H_k as

$$H_k = \begin{bmatrix} I^{(1)} & 0 \\ 0 & H_k^{(2)} \end{bmatrix}, \tag{43}$$

the following results

$$H_k \underline{x} = \begin{bmatrix} I^{(1)} \underline{x}^{(1)} \\ H_k^{(2)} \underline{x}^{(2)} \end{bmatrix} = \begin{bmatrix} \underline{x}^{(1)} \\ H_k^{(2)} \underline{x}^{(2)} \end{bmatrix} = \begin{bmatrix} x_1 & \cdots & x_{k-1} & \left(\sum_{i=k}^{m} x_i^2 \right)^{1/2} & 0 & \cdots & 0 \end{bmatrix}^T . \tag{44}$$

The Householder matrix H_k defined in (43) is an orthogonal matrix that can be written as

$$H_k = I - \frac{1}{\beta_k} \underline{v}_k \underline{v}_k^T \tag{45}$$

and acts like an identity matrix on the first $k-1$ coordinates of any vector $\underline{x} \in \mathbb{R}^m$ and transforms the rest into a unit vector. Moreover, it is not necessary to store the entire matrix H_k, but it is enough to store the $(m-k+1)\times 1$ vector $\underline{v}_k$ and the scalar β_k.

2. QR Decomposition

The QR decomposition is used in order to factor a matrix A into a product of two matrices Q and R [28], [29]

$$A = QR \ . \tag{46}$$

If A is an $m \times n$ matrix, then Q is an $m \times m$ orthogonal matrix and R is an $m \times n$ upper triangular matrix. The QR decomposition is used to solve a linear system $A\underline{x} = \underline{b}$, and it can be computed by applying a series of Householder transformations to matrix A.

By defining

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \tag{47}$$

and using the procedure described in the previous section, an $m \times m$ Householder matrix H_1 can be found, which when applied to the first column of A will give a multiple of $\underline{e}_1$. Thus, the result of multiplying H_1 with A will be

$$H_1 A = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & a_{m2}^{(1)} & a_{m3}^{(1)} & \cdots & a_{mn}^{(1)} \end{bmatrix} \tag{48}$$

where the superscript denotes that the matrix element is affected by the H_1 transformation. Note that H_1 is fully described by vector $\underline{v}_1$ and scalar β_1 as mentioned before. Then, a second Householder transformation H_2 that will zero out the last $n-2$ elements in the second column of $H_1 A$ can be again constructed. Thus, $H_2 H_1 A$ will be of the form

$$H_2 H_1 A = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & a_{m3}^{(2)} & \cdots & a_{mn}^{(2)} \end{bmatrix} \tag{49}$$

where again the superscript indicates the last transformation that affected the corresponding element. For example, the first column and the first row are not affected by the H_2 transformation. From (41), H_2 is of the form

$$H_2 = \begin{bmatrix} I_1 & 0 \\ 0 & H_2^{(2)} \end{bmatrix} \tag{50}$$

where I_1 is a 1×1 identity matrix, and $H_2^{(2)}$ is an $(m-1) \times (m-1)$ Householder matrix. Since $H_2^{(2)}$ is fully determined by an $(m-1) \times 1$ vector $\underline{v}_2^{(2)}$ and a scalar β_2, only m elements are needed to be stored for $H_2^{(2)}$.

Continuing to apply the Householder transformation in this way will result in an upper triangular matrix R, i.e.,

$$H_n H_{n-1} \cdots H_2 H_1 A = R \tag{51}$$

where R is of the form

$$R = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & a_{nn}^{(n)} \\ \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{52}$$

The $m \times n$ matrix R is composed of two matrices, a $n \times n$ upper triangular R_1 and a $(m-n) \times n$ zero matrix:

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \tag{53}$$

Let Q^T, an $m \times m$ matrix, be the product of the Householder transformations

$$Q^T = H_n H_{n-1} \cdots H_2 H_1 = \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \tag{54}$$

where Q_1^T is an $n \times m$ matrix containing the first n rows of Q^T, and Q_2^T is an $(m-n) \times m$ matrix containing the last $m-n$ rows. Since $Q^T A = R$ and Q is orthogonal, it follows that

$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1 . \tag{55}$$

The above derived QR decomposition can be used in order to find the solution to the least squares problem $A\underline{x} = \underline{b}$, where A is an $m \times n$ matrix and $\underline{x}$ and $\underline{b}$ are $n \times 1$ and $m \times 1$ vectors, respectively. The solution $\hat{\underline{x}}$ satisfies the normal equations [29]:

$$A^T A\underline{x} = A^T \underline{b} \tag{56}$$

and by using the QR decomposition, they can be written in the form

$$R_1^T Q_1^T Q_1 R_1 \underline{x} = R_1^T Q_1^T \underline{b} \ . \tag{57}$$

By setting

$$\underline{c} = Q^T \underline{b} = \begin{bmatrix} Q_1^T \underline{b} \\ Q_2^T \underline{b} \end{bmatrix} = \begin{bmatrix} \underline{c_1} \\ \underline{c_2} \end{bmatrix} \tag{58}$$

and taking into account that $Q_1^T Q_1 = I$ and R_1^T is nonsingular, (57) simplifies to

$$R_1 \underline{x} = \underline{c}_1 \ . \tag{59}$$

The system of (59) can be solved using back substitution [29] and the solution

$$\hat{\underline{x}} = R_1^{-1} \underline{c}_1 \tag{60}$$

is the unique solution to the least squares problem.

In summary, if A is an $m \times n$ matrix, the least squares problem can be solved using the QR decomposition by Householder factorizations as follows:

a) Compute $R = H_n H_{n-1} \cdots H_2 H_1 A$ and $\underline{c} = H_n H_{n-1} \cdots H_2 H_1 \underline{b}$.

b) Partition R and $\underline{c}$ into a block form $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ and $\underline{c} = \begin{bmatrix} \underline{c_1} \\ \underline{c_2} \end{bmatrix}$ where R_1 and $\underline{c}_1$ each have n rows.

c) Finally, solve $R_1 \underline{x} = \underline{c}_1$ using back substitution.

3. Proposed Algorithm Description

In Chapter III, a method for determining the weights that provide the best approximation of a desired response in the LS sense was described. The LS problem of (30) can be solved using the QR factorization presented in the previous section. The cluster head collects all the positions from the sensor nodes and constructs the steering matrix $D(\theta)^H$. Then it computes the QR factorization using Householder transformations and finds the solution of the weight vector $\underline{w}$ for a given desired response $\underline{F}_d(\theta)$. The procedure is straightforward, but it lacks robustness since a single node bears the entire processing load, which increases the possibility of failure, causing the problem to be solved again from scratch. Therefore, a distributed algorithm for solving the LS problem with the QR decomposition is desirable. The processing load is shared by the nodes, but there is also a tradeoff of increased communication load.

For the implementation of the algorithm, the following reasonable assumptions are made:

a) Each node can calculate its position from a reference node accurately.

b) Each node can broadcast information to other nodes in the cluster.

c) The desired array response for a set of m directions $\theta_1,...,\theta_m$ of the incoming signal is known.

d) Errors due to the noise during the communication among the nodes are not taken into account.

The algorithm exploits the specific nature of the matrix $D(\theta)^H$ to be decomposed by the QR factorization. By taking the transpose of (28), the steering matrix is

$$D^H(\theta) = \begin{bmatrix} e^{j\beta x_1 \sin\theta_1} & e^{j\beta x_2 \sin\theta_1} & \cdots & e^{j\beta x_n \sin\theta_1} \\ e^{j\beta x_1 \sin\theta_2} & e^{j\beta x_2 \sin\theta_2} & \cdots & e^{j\beta x_n \sin\theta_2} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\beta x_1 \sin\theta_m} & e^{j\beta x_2 \sin\theta_m} & \cdots & e^{j\beta x_n \sin\theta_m} \end{bmatrix} \tag{61}$$

where n is the number of sensor nodes and m the number of angles or approximation points. Obviously, the first column is constructed by the position of the first node x_1 and the set of angles $\theta_1,...,\theta_m$. Similarly, the elements of the second column depend on the second node position x_2 and the set of m angles. In general, each column i of $D^H(\theta)$ depends only on the position x_i of the corresponding node and the desired angles.

From (61), it is obvious that since one node knows its position from a reference node and the set of m angles, it can construct its corresponding column of $D^H(\theta)$ without exchanging any information, such as their position, with the other nodes. Thus, the matrix $D^H(\theta)$ can be stored in a distributed way among the sensor nodes.

From the analysis of the previous section comes the observation that the first Householder transformation H_1 is needed to zero out all the elements of the first column except for the first element. Since for the construction of matrix H_1 only the data values from the first column are used, the conclusion drawn is that it can be computed by the first node only. Therefore, the first node, which initializes the QR decomposition, calculates the H_1 matrix and multiplies its own first column by this matrix. Then it broadcasts H_1 to all other nodes, which in turn transform their corresponding columns with H_1. There is no need to transmit the entire matrix H_1 but only the vector $\underline{v}_1$ and the scalar β_1. After this first step of H_1 computation, transmission and application is completed, the distributed steering matrix will have the form of (48):

$$D^H_{(1)}(\theta) = H_1 D^H(\theta) = \begin{bmatrix} a^{(1)}_{11} & a^{(1)}_{12} & a^{(1)}_{13} & \cdots & a^{(1)}_{1n} \\ 0 & a^{(1)}_{22} & a^{(1)}_{23} & \cdots & a^{(1)}_{2n} \\ 0 & a^{(1)}_{32} & a^{(1)}_{33} & \cdots & a^{(1)}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & a^{(1)}_{m2} & a^{(1)}_{m3} & \cdots & a^{(1)}_{mn} \end{bmatrix} \tag{62}$$

where $a^{(1)}_{ij}$ are the matrix elements transformed by H_1.

The second step is similar to the first and is performed by the second node. The second node computes the H_2 transformation, which depends only on the second

column, and then H_2 (vector $\underline{v}_2$ and scalar β_2) is broadcast to each node which in turn applies the new Householder transformation to its column. The resulting distributed matrix is

$$D_{(2)}^{H}(\theta) = H_2 H_1 D^{H}(\theta) = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & a_{m3}^{(2)} & \cdots & a_{mn}^{(2)} \end{bmatrix}. \tag{63}$$

From the structure of $D_{(2)}^{H}(\theta)$, the following observations are made. The first node receives the H_2 transformation, but it does not need to apply it to its column since H_2 does not affect the first column. Additionally, each node does not need to apply the H_2 to the first element of its column since the first row is not affected by the H_2 transformation. Similarly, the following Householder transformations are applied only when needed, thus avoiding redundant calculations.

The procedure goes on until all Householder transformations $H_1,\ldots,H_n$ are computed, broadcast and applied by the nodes. The distributed matrix will result in an upper triangular form

$$D_{(n)}^{H}(\theta) = H_n H_{n-1} \cdots H_2 H_1 D^{H}(\theta) = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & a_{nn}^{(n)} \\ \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{64}$$

The QR decomposition of the $D^{H}(\theta)$ matrix is achieved in a fully distributed way, and the processing load is shared among the nodes. There is no additional processing effort since the Householder transformations are applied only to the affected elements, as

described before. There is no difference between the centralized and the distributed QR decomposition in terms of the processing load. The significant advantage is that instead of one node bearing all the computational burden, each node in the array takes its turn in completing the QR decomposition. However, there is some additional communication load since the H_i matrices need to be broadcast. This is the tradeoff for relieving the cluster head from the heavy processing load. This first phase of the algorithm is depicted in Figure 17. In the 1st step the 1st node calculates the matrix H_1 and broadcasts it to all the other sensors. In the 2nd step the 2nd node calculate the matrix H_2 and the procedure continues until the last node which computes the last matrix H_3

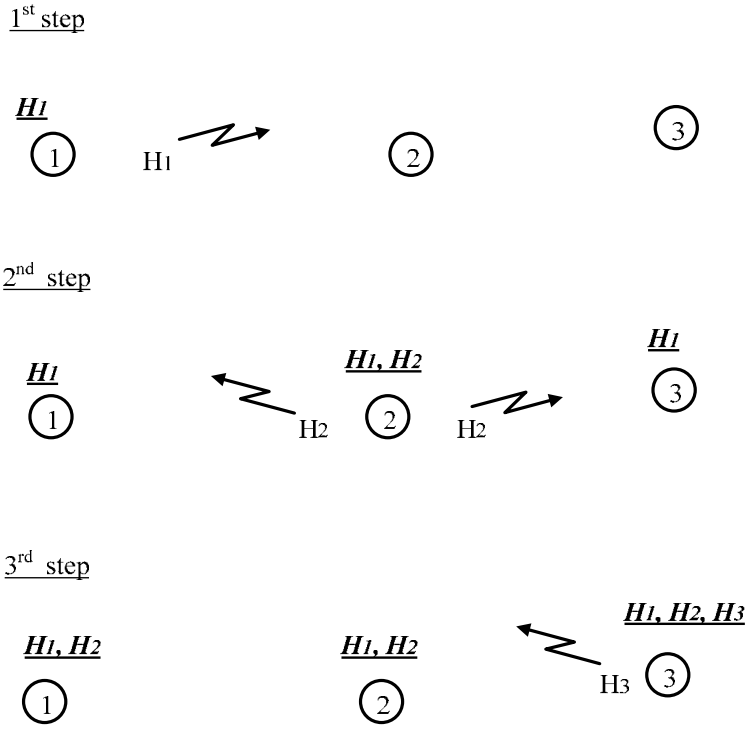


Figure 17. First phase of the algorithm for distributed QR decomposition by the sensor nodes. Bolded and underlined H_i above the nodes denote the Householder transformations that are already stored in the sensor. Simple H_i denote the just computed and broadcast Householder transformation.

The next phase in solving the LS problem is to update the desired response using a series of Householder transformations to obtain vector $\underline{c}$:

$$\underline{c} = H_n H_{n-1} \cdots H_2 H_1 \underline{F}_d(\theta) = \begin{bmatrix} F_d^{(1)}(\theta_1) \\ F_d^{(2)}(\theta_2) \\ F_d^{(3)}(\theta_3) \\ \vdots \\ F_d^{(n)}(\theta_n) \\ \vdots \\ F_d^{(n)}(\theta_m) \end{bmatrix} \tag{65}$$

which is then partitioned to $\underline{c} = \begin{bmatrix} \underline{c}_1 \\ \underline{c}_2 \end{bmatrix}$ according to (58).

The third phase includes the solution of the system

$$R_1 \underline{w} = \underline{c}_1 = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{66}$$

by back substitution where R_1 defined by (52) and (53) is an upper triangular $n \times n$ matrix containing the first n rows of $D_{(n)}^H(\theta)$:

$$R_1 = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} . \tag{67}$$

The n^{th} node needs to solve the last equation of the system

$$a_{nn}^{(n)} w_n = c_n \tag{68}$$

and immediately computes its weight w_n for beamforming. The $(n-1)^{th}$ node needs to solve the $(n-1)^{th}$ equation in order to find its weight w_{n-1}:

$$a_{n-1,n-1}^{(n-1)} w_{n-1} + a_{n-1,n}^{(n-1)} w_n = c_{n-1} . \tag{69}$$

The $a_{n-1,n-1}^{(n-1)}$ data element of matrix R_1 in (67) is constructed by the $(n-1)^{th}$ node by applying all transformations H_1 to H_{n-1} to the initial element $a_{n-1,n-1} = e^{j\beta x_{n-1}\sin\theta_{n-1}}$ since this data element belongs to its own $(n-1)^{th}$ column. However, it needs the value $a_{n-1,n}^{(n-1)}$, which is in the n^{th} column; thus, it depends on the position x_n of the n^{th} node. It also needs to receive the weight w_n and compute the value c_{n-1}. The n^{th} node broadcasts its position x_n and its weight w_n to all nodes. Now, the $(n-1)^{th}$ node can solve (69) and calculate the weight w_{n-1}.

Similarly, the $(n-1)^{th}$ node broadcasts its position x_{n-1} and its weight w_{n-1} to all the other nodes, which need this information to solve their own equation. For example, the $(n-2)^{th}$ node has to solve the equation

$$a_{n-2,n-2}^{(n-2)} w_{n-2} + a_{n-2,n-1}^{(n-2)} w_{n-1} + a_{n-2,n}^{(n-2)} w_n = c_{n-2}, \tag{70}$$

so it needs the positions x_n and x_{n-1} in order to construct the $a_{n-2,n-1}^{(n-2)}$ and $a_{n-2,n}^{(n-2)}$ elements, respectively, by applying all the Householder transformations up to H_{n-2}. It also needs the weights w_{n-1} and w_n from the previous nodes.

The final result is that every node obtains its own weight, and the sensor network is now ready to cooperate in order to form an antenna array. The third phase of the algorithm is depicted in Figure 18. During the 1st step the last node (3rd), calculates its own weight w_3 and broadcasts it along with its position x_3. In the 2nd step the 2nd node uses the received information to solve for its weight w_2. Then it broadcasts w_2 and its position x_2. The procedure goes on until the 1st node calculates its own weight w_1

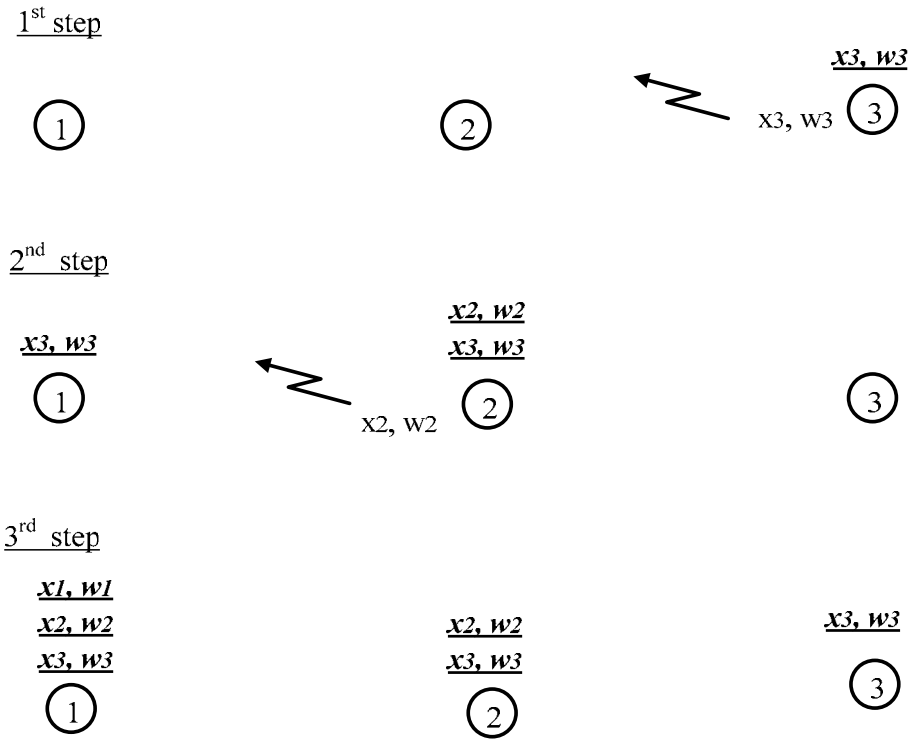


Figure 18. Last phase of the algorithm to implement distributed back substitution by the sensor nodes. Bolded and underlined x_i, w_i above the nodes denote the positions and weights that are already stored in the sensor. Simple x_i, w_i denote the broadcast position and just computed weight.

4. Computational and Communication Cost Analysis

The computational cost of the procedure presented in the previous section can be measured in number of instructions needed for the implementation. It is known that the number of flops for the solution of the LS problem in a central processor is given by [28]

$$N_i = 2n^2(m - n/3) + mn + n^2 \tag{71}$$

where the first term is for the determination of the QR factorization, the second term for the update of the desired response vector with the Householder transformations, and the last term accounts for the back substitution to solve for the weights.

The significant characteristic of the proposed distributed algorithm is that, as described before, no redundant computations are made. The Householder transformations are calculated in exactly the same way as the centralized approach, which yields the same

number of computations. In the centralized approach, however, each element of the distributed stored steering matrix $D_{(n)}^{H}(\theta)$, or similarly the upper triangular matrix R_1 of (67), is computed by a single sensor and no redundant computations are made. For example, in the distributed approach, the first node does not construct the entire matrix. In the first phase, it uses only the first column for defining the first Householder matrix H_1, and then in the back substitution phase it calculates only the first row of the R_1 using the H_1. Similarly, the second node calculates H_2 using the second column and updates the second row, which is used for the back substitution. The following matrix shows the elements of R_1 with respect to the node that uses and calculates them:

$$R_1 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 2 & 2 & \cdots & 2 \\ \vdots & 0 & 3 & \cdots & 3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & n \end{bmatrix} \tag{72}$$

Therefore, no additional computations are made, and the total processing cost is exactly equal to that of the centralized approach given in (71). Thus, without any increase, the processing load is shared among the nodes, and the cluster head is relieved from the heavy computational effort.

The number of instructions N_i can be used to express the processing power using the definitions of Chapter II for computational cost. Therefore, from (19) and (71), the processing power P_p is given by

$$P_p = N_i \times P_i = \left[2n^2(m - n/3) + mn + n^2 \right] \times P_i \ . \tag{73}$$

However, this distribution of the processing load comes with a tradeoff, which is an increase in the communication load. Note that throughout this work the communication cost is only the number of data elements that need to be transmitted for the implementation of an algorithm. The transmissions required for the coordination of the sensors in the distributed algorithm are not taken into account for the calculation of the communication cost. In the centralized approach, the cluster head collects all position

data from n nodes and sends back n weights, so the number of transmitted data elements N_t is equal to

$$N_t = 2n . \tag{74}$$

On the other hand, in the distributed algorithm, the number of transmitted data elements is calculated as follows. The first node sends the H_1 matrix, or equivalently the $m \times 1$ vector $\underline{v}_1$ and the scalar β_1, a total of $m+1$ data elements. Similarly, the second node broadcasts the matrix H_2 or the $(m-1)\times 1$ vector $\underline{v}_2$ and the scalar β_2, a total of m data elements. This procedure is continued until the $(n-1)^{th}$ node, which transmits the H_{n-1}. In general, an arbitrary node i transmits the matrix H_i, i.e., $(m-i+1)$ elements for $\underline{v}_i$ and one for β_i. The total number of transmitted elements by all nodes for the first phase of the QR decomposition is given by

$$C_{QR} = \sum_{i=1}^{n-1}(m-1+2) = (m+2)(n-1) - \frac{n(n-1)}{2} = (m+2-n/2)(n-1) . \tag{75}$$

For the phase of the back substitution, the n^{th} node transmits its position and its weight, a total of two elements. Similarly, each node except the first one, broadcasts its position and weight to the other nodes. Thus, the total number transmitted elements for the back substitution is given by

$$C_{BS} = 2(n-1) . \tag{76}$$

Finally, the total number of data elements to be broadcast during the implementation of the algorithm is

$$C = C_{QR} + C_{BS} = (m+4-n/2)(n-1) . \tag{77}$$

Using again the definitions from Chapter II for the communication power, the power consumption due to communication can be derived. Assuming that each element is represented by b bits, the number of transmitted bits is

$$N_{tb} = C \times b \tag{78}$$

and from (18) the communication power P_c is given by

$$P_c = N_{tb} \times P_{tb} = (m + 4 - n/2)(n-1) \times P_{tb} \times b\,. \tag{79}$$

Therefore, the total power for the implementation of the algorithm is the sum of (73) and (79)

$$P = \left[2n^2 (m - n/3) + mn + n^2 \right] \times P_i + (m + 4 - n/2)(n-1) \times P_{tb} \times b \tag{80}$$

and depends highly on the specific characteristics P_i, P_{tb} and b of the sensors. Similarly, the total power for the centralized implementation is calculated to be

$$P = \left[2n^2 (m - n/3) + mn + n^2 \right] \times P_i + 2n \times P_{tb} \times b\,. \tag{81}$$

The following figures summarize the results and compare the two implementations.

In Figure 19, the number of required instructions N_i is plotted as a function of the number of approximation points m for a 10×1 array of sensors while in Figure 20, N_i is plotted as a function of the number of sensors n, for $m = 20$ approximation points. The plots for the processing power P_p can be obtained by multiplying the number of instructions with P_i as in (73). Another important point is that the main processing effort happens during the QR decomposition phase as the back substitution procedure is not so demanding. This is significant because the QR decomposition is computed once and then the results are kept for future modifications. For example, if a new array response is required, it is only necessary to compute the updated vector $\underline{c}_1$ of (65) and then solve the system of (66) by back substitution; R_1 does not change. Therefore, in the case of communication with a UAV, which is moving and causing the desired response to change continuously, the sensor array only has to quickly perform the back substitution phase in order to compute the new weights.

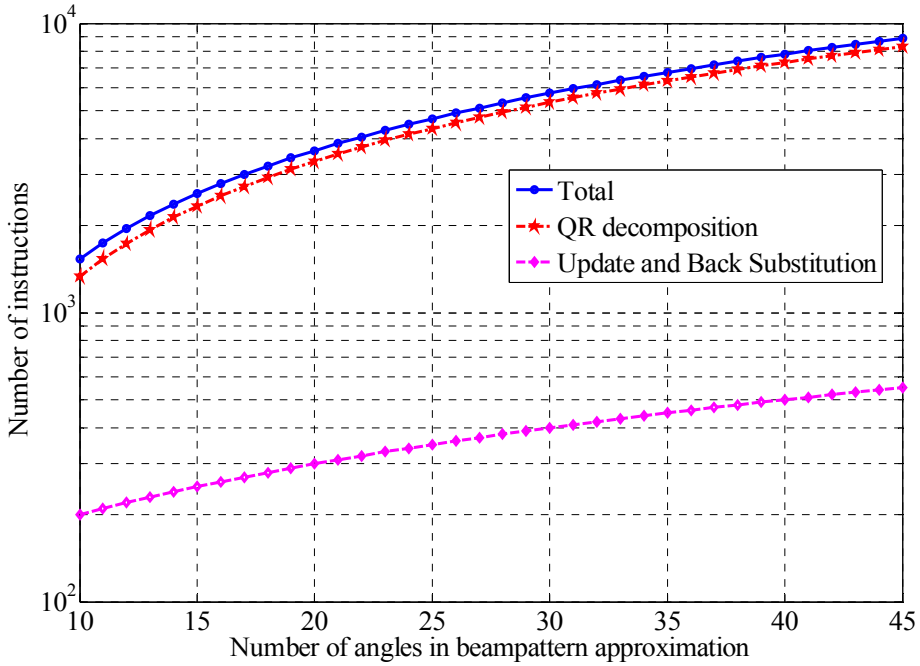


Figure 19. Processing cost of the distributed algorithm as a function of the number of approximation angles and for $n = 10$ sensors. Multiplying N_i by P_i gives the required processing power.

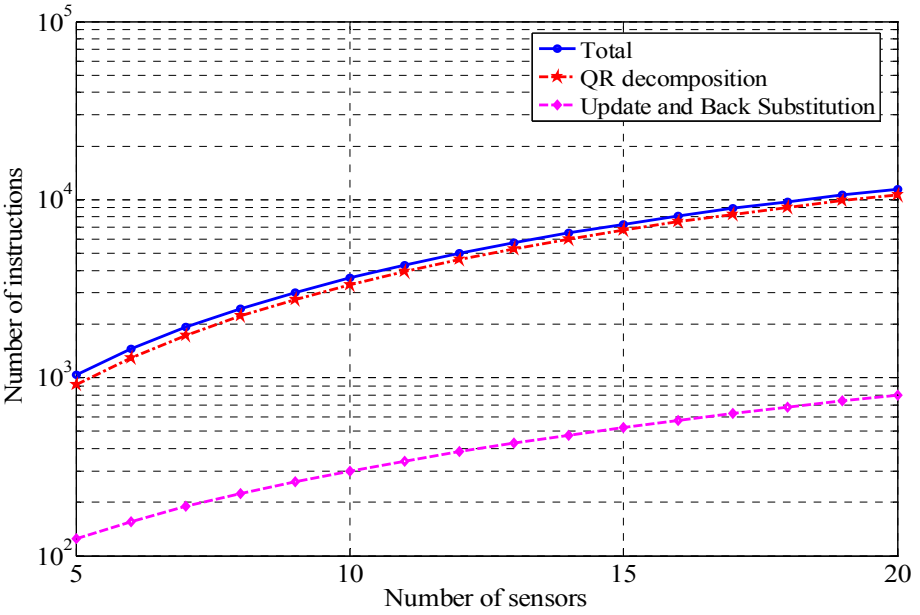


Figure 20. Processing cost of the distributed algorithm as a function of the number of sensors and for fixed number of approximation angles $m = 20$. Multiplying N_i by P_i gives the required processing power.

The communication cost as a function of the number of approximation points and the number of sensors is depicted in Figures 21 and 22, respectively. Figure 21 shows the communication cost for a 10×1 sensor array in terms of the transmitted elements. Assuming that each data element is represented by $b=32$ bits in a single precision floating point representation, the total transmission power can be found by multiplying the number of elements C by $P_{tb}\times b$ as in (78) and (79). Similarly, in Figure 22, the number of transmitted elements C is plotted as a function of the number of sensors n for $m=20$. Compared to the centralized approach, the communication load is increased, especially due to the first phase of the QR decomposition. However, as described before for the processing load, this phase has to be implemented only once. Therefore, any change in the desired response needs only the back substitution phase, which requires a considerably lower number of elements to be transmitted.

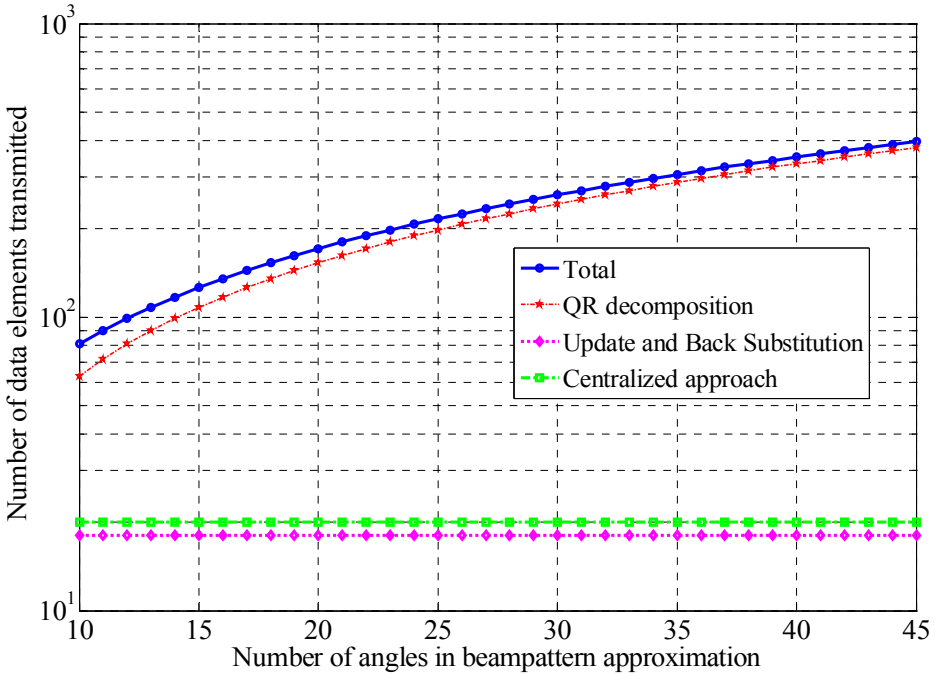


Figure 21. Communication cost of the distributed algorithm as a function of the number of approximation angles and for a fixed number of sensors ($n=10$). Multiplying the number of data elements with $P_{tb}\times b$ gives the required transmission power.

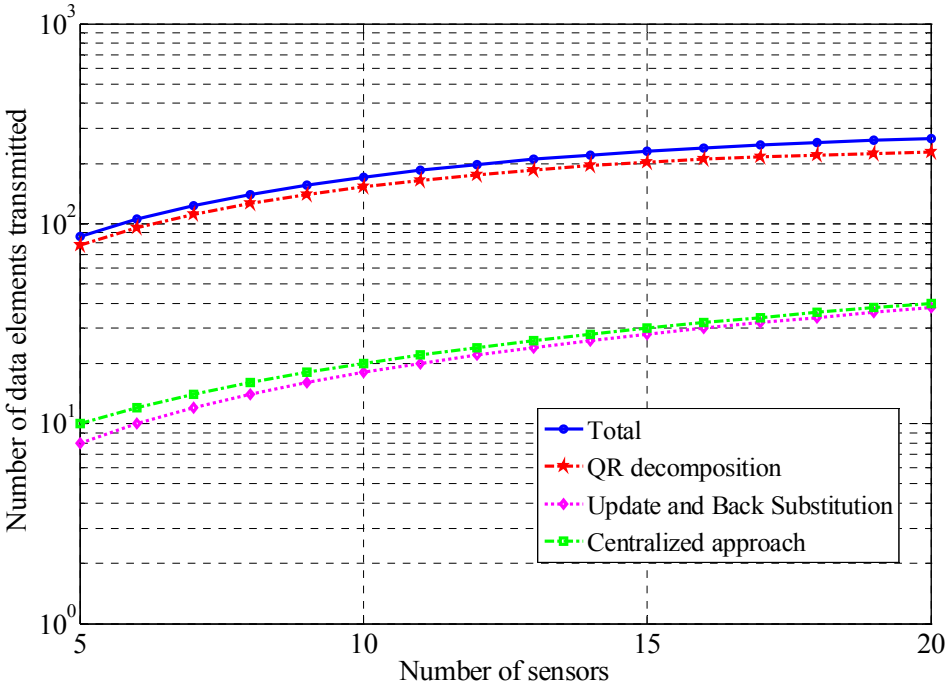


Figure 22. Communication cost of the distributed algorithm as a function of the number of sensors and for fixed number of approximation angles ($m = 20$). Multiplying the number of data elements with $P_{tb} \times b$ gives the required transmission power.

In Figures 23 and 24, the total power consumption as a function of the number of approximation points and the number of sensors, respectively, is plotted. Assuming that P_i is equal to one unit of power, and using the ratio η_{tp}, the factor P_{tb} can be substituted by $P_{tb} = \eta_{tp} \times P_i$. Dividing the total power P of (80) by P_i yields the normalized power P_n in terms of the required units of power P_i:

$$P_n = \frac{P}{P_i} = \left[2n^2 (m - n/3) + mn + n^2 \right] + (m + 4 - n/2)(n-1) \times b \times \eta_{tp} \qquad (82)$$

From the figures, it is obvious that the distributed algorithm requires more power than the centralized one, and this is caused by the additional communication load. However, this total power cost is shared among the nodes of the sensor network and is not undertaken by a single node. For example, Figure 24 shows that a deployment of twenty sensors needs about five times more power for the distributed approach than the

centralized approach. Therefore, the cluster head in a sensor node consumes four times more power in the centralized scheme than the average sensor nodes consume in the distributed one. This obviously will cause the cluster head to fail quickly, which means that a new cluster head will need to be selected and all the computations from the beginning will need to be re-done. Thus, the increase in the power consumption in the distributed approach can be considered reasonable if the robustness of the network is a crucial requirement.

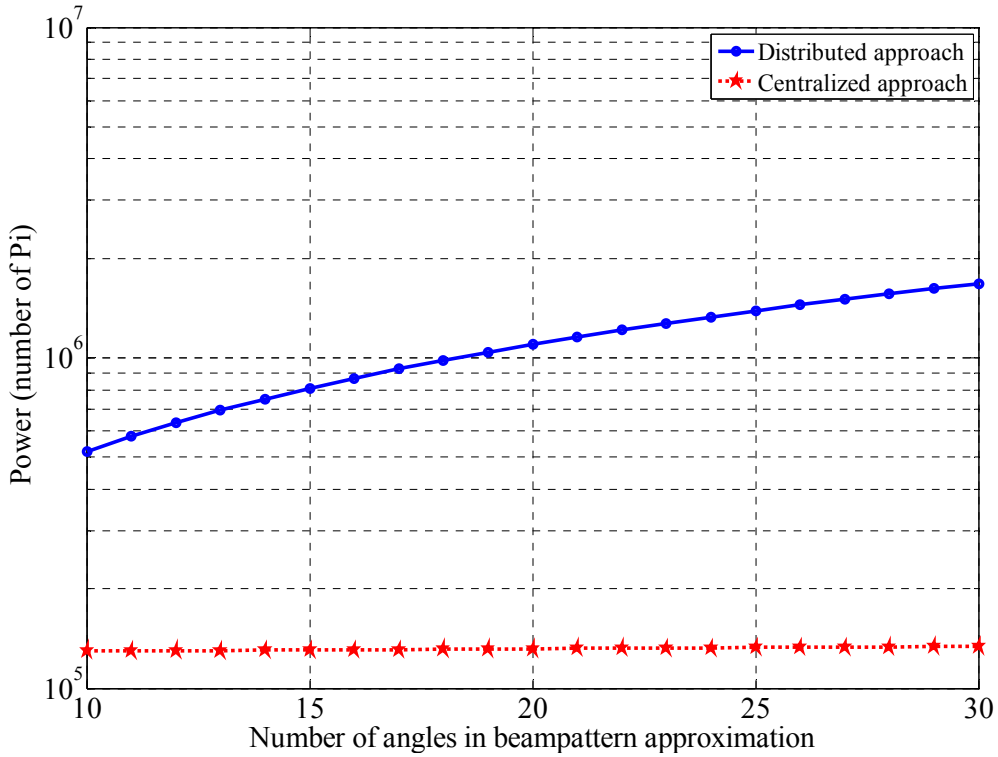


Figure 23. Normalized power P_n (number of P_i) for both distributed and centralized approaches as a function of the number of approximation angles for a fixed number of sensors ($n = 20$). η_{tp} is assumed to be 200 and $b = 32$ bits.

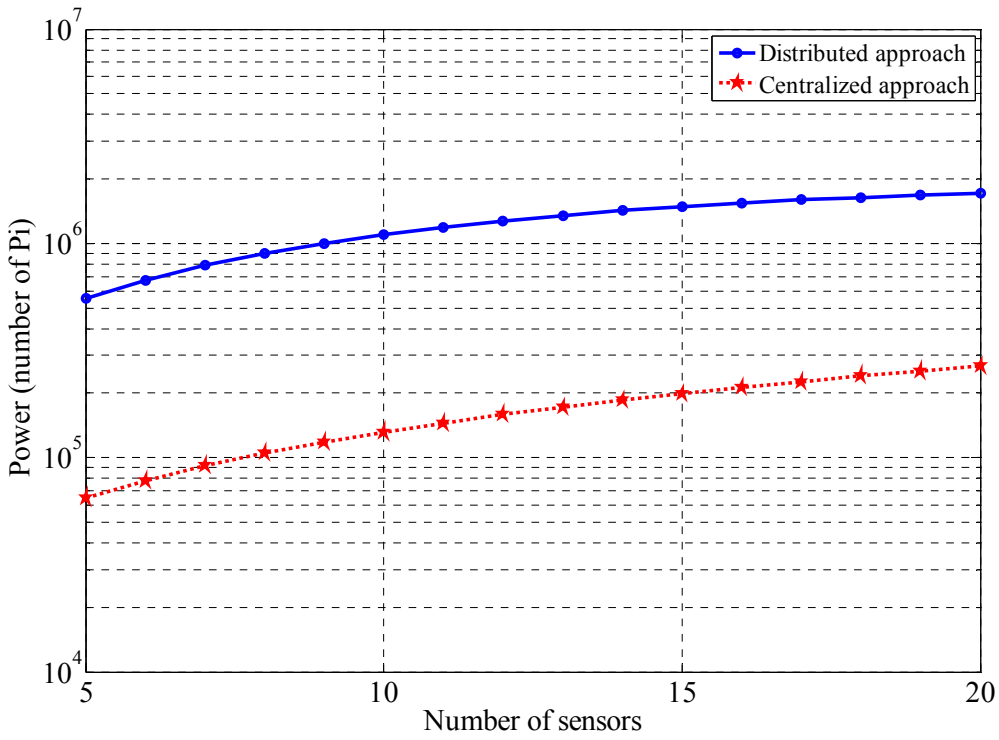


Figure 24. Normalized power P_n (number of P_i) for both distributed and centralized approaches as a function of the number of sensors for a fixed number of approximation angles ($m = 20$). η_{tp} is assumed to be 200 and $b = 32$.

B. DISTRIBUTED ITERATIVE SCHEME FOR SOLVING THE LEAST SQUARE PROBLEM

In this selection, a distributed scheme based on an iterative solution of the LS problem is presented and evaluated. The iterative procedure is performed in steps by all the nodes in the sensor array. Starting from an arbitrary initialization for the weights, this method quickly converges to the actual solution of the LS problem.

For the implementation of the algorithm, the following assumptions are made:

a) Each node can calculate its position from a reference node accurately.

b) Each node can broadcast information to other nodes in the cluster.

c) The desired array response for a set of directions $\theta_1,...,\theta_m$ of the incoming signal is known.

d) Errors due to noise during the communication among the nodes are not taken into account.

The linear system of (31), which is to be solved in the LS sense, is considered a minimization problem of the following form:

$$\varepsilon = \min_{\underline{x}} \left| A\underline{x} - \underline{b} \right|^2 \tag{83}$$

where

$$\begin{aligned} A &\equiv D(\theta)^H \\ \underline{x} &\equiv \underline{w} \\ \underline{b} &\equiv \underline{F}_d(\theta) \ . \end{aligned} \tag{84}$$

1. Proposed Algorithm

The algorithm is based on various parallel methods proposed in the literature for solving the LS problem [30], [31], [32], [33].

Let A be an $m \times n$ matrix while $\underline{x}$ and $\underline{b}$ are $n \times 1$ and $m \times 1$ vectors, respectively. Each column of A is denoted $\underline{A}_i$ and each scalar element of vector $\underline{x}$ is denoted x_i, for $i = 1,\ldots,n$. Then, the LS problem has the equivalent form

$$\varepsilon = \min_{\underline{x}} \left| \underline{A}_1 x_1 + \underline{A}_2 x_2 + \ldots + \underline{A}_n x_n - \underline{b} \right|^2 \ . \tag{85}$$

Suppose that $\underline{x}^{(k)}$ is an approximation to the solution $\underline{x}^*$ after k iterations and its elements are $x_i^{(k)}$, for $i = 1,\ldots,n$. Considering an arbitrary element x_i, all elements $x_1,\ldots x_{i-1}$ are updated in $k+1$ iterations while the rest $x_i,\ldots x_n$ are updated in k iterations. Now, (85) can be written as

$$\varepsilon = \min_{x_i^{(k+1)}} \left| \sum_{j=1}^{i-1} \underline{A}_j x_j^{(k+1)} + \underline{A}_i x_i^{(k+1)} + \sum_{j=i+1}^{n} \underline{A}_j x_j^{(k)} - \underline{b} \right|^2 \tag{86}$$

for the $(k+1)^{th}$ iteration of $x_i^{(k+1)}$, which gives the local solution for $x_i^{(k+1)}$. The argument of the term on the right hand side of (86) can be written as

$$\begin{aligned}&\sum_{j=1}^{i-1}\underline{A}_j x_j^{(k+1)} + \underline{A}_i x_i^{(k+1)} + \sum_{j=i+1}^{n}\underline{A}_j x_j^{(k)} - \underline{b} = \\ &\underline{A}_i(x_i^{(k+1)} - x_i^{(k)}) + \sum_{j=1}^{i-1}\underline{A}_j x_j^{(k+1)} + \sum_{j=i}^{n}\underline{A}_j x_j^{(k)} - \underline{b}\end{aligned} \quad . \tag{87}$$

By substituting $s_i^{(k)}$ as the step or correction

$$s_i^{(k)} = x_i^{(k+1)} - x_i^{(k)} \tag{88}$$

and $\underline{r}^{(k,i-1)}$ as the residual

$$\underline{r}^{(k,i-1)} = \sum_{j=1}^{i-1}\underline{A}_j x_j^{(k+1)} + \sum_{j=i}^{n}\underline{A}_j x_j^{(k)} - \underline{b} \tag{89}$$

into (86) yields

$$\varepsilon = \min_{s_i^{(k)}} \left|\underline{A}_i s_i^{(k)} + \underline{r}^{(k,i-1)}\right|^2 . \tag{90}$$

Thus, the global problem of finding the solution for $x_1,\ldots,x_n$ in (85) is equivalent to solving the subproblems of (90), which can be assigned to the corresponding sensor nodes. Indeed, $\underline{A}_i$ is known locally to the sensor node i, $\underline{s}_i^{(k)}$ is the locally computed correction, and $\underline{r}^{(k,i-1)}$ is the residual after the $i-1^{th}$ node has completed its update $x_{i-1}^{(k)}$. The residual $\underline{r}^{(k,i-1)}$ is not locally available in the node i; however, it can be transmitted by the $i-1^{th}$ sensor node. After the i^{th} node calculates the new solution $x_i^{(k)}$, it sends the updated residual $\underline{r}^{(k,i)}$ to the other nodes. The residual can be shown to satisfy the recursive equation [30]

$$\underline{r}^{(k,i)} = \underline{r}^{(k,i-1)} + \underline{A}_i s_i^{(k)} \tag{91}$$

whereas the new approximate solution is

$$x_i^{(k+1)} = x_i^{(k)} + s_i^{(k)} . \tag{92}$$

Therefore, the i^{th} node is assigned a column $\underline{A}_i$ of the matrix A and assumes an initial solution $x_i^{(0)}$. Assuming that $\underline{r}^{(0)}$, the initial estimation for the residual is available, the i^{th} node solves (90) for $s_i^{(1)}$ and then updates its solution $x_i^{(1)}$ by (92). Following this step, the updated residual $\underline{r}^{(1,i)}$ is sent to the $i+1^{th}$ node in order to update its solution $x_{i+1}^{(1)}$. The procedure continues until a convergence criterion is satisfied. This series of approximations converge to the solution $\underline{x}^*$, and the norm of the residual decreases continuously [30], [31]. This procedure is summarized in Figure 25.

a) Divide A into its columns $\underline{A}_i$, for $i = 1,\ldots,n,$ and assign one to each node i.

b) Initialize $x_i^{(0)}$, for $i = 1,\ldots,n,$ and $\underline{r}^{(0)}$.

c) For $k = 1$ until convergence

For $i = 1,\ldots,n$

Solve for $s_i^{(k)}$: $\min_{s_i^{(k)}} \left| \underline{A}_i s_i^{(k)} + \underline{r}^{(k,i-1)} \right|^2$.

Update the residual : $\underline{r}^{(k,i)} = \underline{r}^{(k,i-1)} + \underline{A}_i s_i^{(k)}$.

Send the updated residual to all nodes.

Update the solution $x_i^{(k+1)} = x_i^{(k)} + s_i^{(k)}$.

Check for convergence.

Figure 25. Procedure for the distributed iterative solution of the LS problem (After Ref. [30]).

This algorithm is highly distributed, and it can be implemented in a sensor network in order to spread the computational effort equally among all participating sensor nodes.

From the notation of (84), the matrix A is the steering matrix $D(\theta)^H$, each column $\underline{A}_i$ is the i^{th} column of $D(\theta)^H$, and x_i is the weight coefficient w_i. From (61), each column of $D(\theta)^H$ depends only on the position of the node and the set of approximation points. This column is available locally to the sensor node. Starting from an initial estimation for the residual $\underline{r}^{(0)}$, the first node can solve (90) for $s_1^{(1)}$ and update its own weight $w_1^{(1)}$ by (92). Then, it can update the residual $\underline{r}^{(1,1)}$ and send it to the next node. However, since the residual is an $m \times 1$ vector, each transmission of the residual needs the transmission of m elements, which is a considerable amount of communication load. Another approach for the transmission of the residual seems more efficient. Assume that all sensors' positions are initially broadcast to all sensor nodes so that each node knows the position of each other node; in this way, they can construct any of the columns of the steering matrix. Then the i^{th} sensor node does not need to send the entire residual $\underline{r}^{(k,i)}$ but only the scalar correction $s_i^{(k)}$, and all other nodes can reconstruct the residual by repeating the (91). For example, the second node is not required to send the residual $\underline{r}^{(k,2)}$ but just the scalar correction $s_2^{(k)}$, and then the third node will reconstruct the residual by computing (91):

$$\underline{r}^{(k,2)} = \underline{r}^{(k,1)} + \underline{A}_2 s_2^{(k)}. \tag{93}$$

Therefore, in each iteration step, only one scalar element has to be broadcast, and the rest of the computations are performed locally. This distributed determination of the weights in a sensor network is summarized in Figure 26.

a) All nodes broadcast their position; each node can construct all columns $\underline{D}_i$, for $i = 1,\ldots,n$, of $D(\theta)^H$.

b) Initialize $w_i^{(0)}$, $i = 1,\ldots,n$ and $\underline{r}^{(0)}$.

c) For $k = 1$ until convergence

For $i = 1,\ldots,n$

Receive $s_{i-1}^{(k)}$ from $i-1$ node.

Reconstruct the residual: $\underline{r}^{(k,i-1)} = \underline{r}^{(k,i-2)} + \underline{D}_{i-1} s_{i-1}^{(k)}$.

Solve for $s_i^{(k)}$: $\min_{s_i^{(k)}} \left| \underline{D}_i s_i^{(k)} + \underline{r}^{(k,i-1)} \right|^2$.

Update the residual: $\underline{r}^{(k,i)} = \underline{r}^{(k,i-1)} + \underline{D}_i s_i^{(k)}$.

Send the only the correction $s_i^{(k)}$ to all nodes.

Update the solution $w_i^{(k+1)} = w_i^{(k)} + s_i^{(k)}$.

Check for convergence.

Figure 26. Procedure for the proposed distributed iterative solution of the LS problem in a WSN environment.

2. Computational and Communication Costs

The performance of the distributed iterative procedure for the computation of the weights in a random sensor array is shown in Figures 27 and 28. In Figure 27, the residual norm is plotted for each local iteration for a 10×1 array of sensors. It is obvious that the residual norm converges to the actual residual after about thirty local iterations or about three complete iterations; one complete iteration is finished when all ten nodes perform a local iteration. Similar results are plotted in Figure 28 where the norm of the

error $e(k)$ between the approximate $\underline{w}^{(k)}$ and the actual solution $\underline{w}^{*}$ calculated after each complete iteration k is

$$e(k) = \sqrt{\left|\underline{w}^{(k)} - \underline{w}^{*}\right|^2} \,. \tag{94}$$

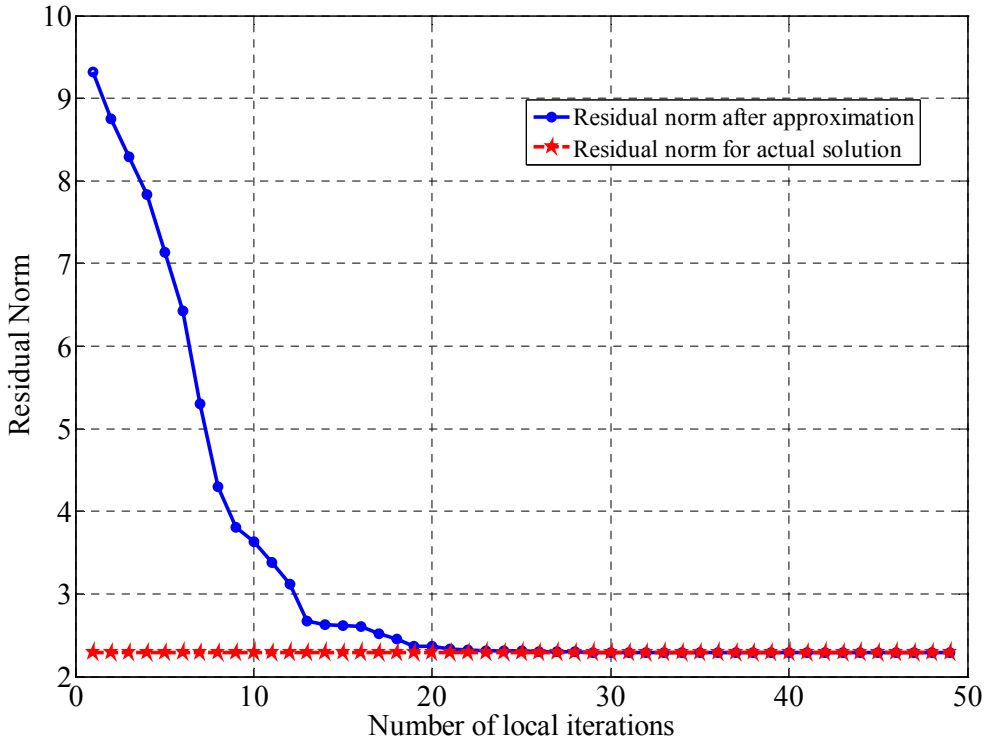


Figure 27. Convergence of the residual norm to the actual residual, indicating that the algorithm converges to the real solution. After 3 complete iterations (30 local) the residual has converged.

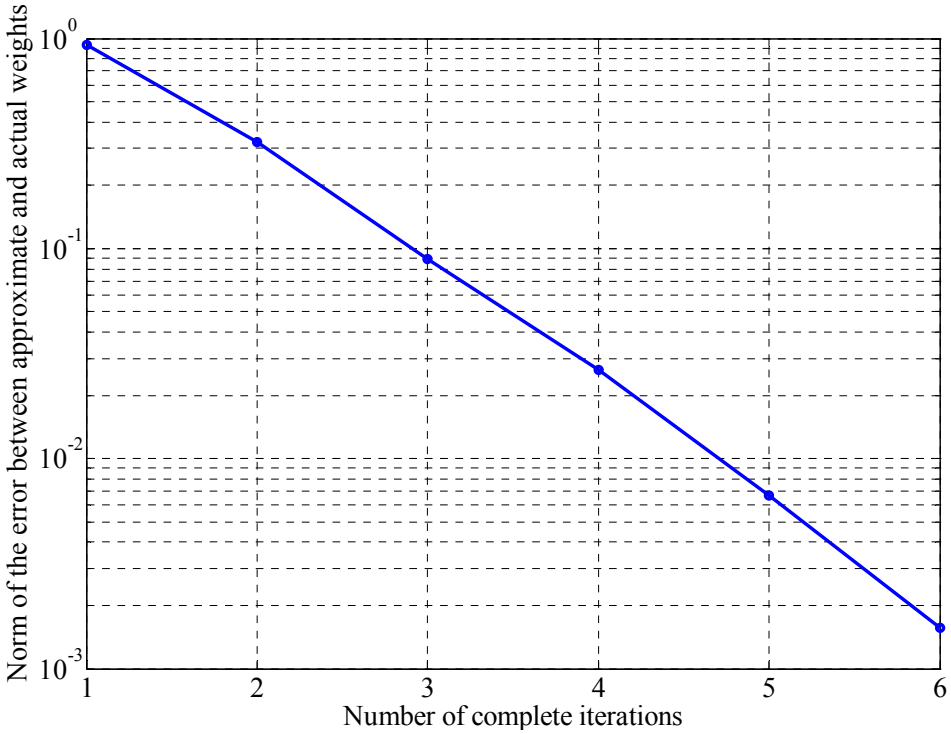


Figure 28. Convergence of the norm of the error $e(k)$ between the approximate $\underline{w}^{(k)}$ and actual solution $\underline{w}^{*}$ calculated after each complete iteration k, indicating that the algorithm converges to the real solution.

The computational cost can be measured in terms of the number of instructions needed for the implementation. Assuming that each node solves its local LS problem with a QR factorization obtained by Householder transformations, the cost per sensor N_{ps} is given by (71), which for $n = 1$ yields

$$N_{ps} = 2(m - 1/3) + m + 1 \ . \tag{95}$$

Note that the last two terms stand for the backsubstitution, so they are performed in each iteration, thus (95) becomes

$$N_{ps} = 2(m - 1/3) + k(m + 1) \, . \tag{96}$$

Furthermore, each node has to reconstruct the $m \times 1$ residual vector, so an additional number of $2m$ instructions are needed. Finally, the number of instructions per sensor results in

$$N_{ps} = 2(m - 1/3) + k(3m + 1)\,. \tag{97}$$

Therefore, the total processing cost for all the nodes N_i is equal to

$$N_i = n \times N_{ps} = n\left[2(m - 1/3) + k(3m + 1)\right] \tag{98}$$

while the total processing power is

$$P_p = N_i \times P_i = n\left[2(m - 1/3) + k(3m + 1)\right] \times P_i\,. \tag{99}$$

Using the above equation, the processing cost in number of instructions for the implementation of the algorithm is plotted as a function of the number of sensors n, the number of approximation angles m, and the number of iterations k. In Figure 29, the number of instructions for the distributed approach become fewer than the centralized approach after a certain point, which is $m = 15$; the simulation scenario is for a 10×1 array of sensors and the cost has been computed for $k = 5$ iterations. Similarly, in Figure 30, the processing cost for the centralized approach grows larger than the distributed approach as the number of the sensors increase ($m = 20$ and $k = 5$).

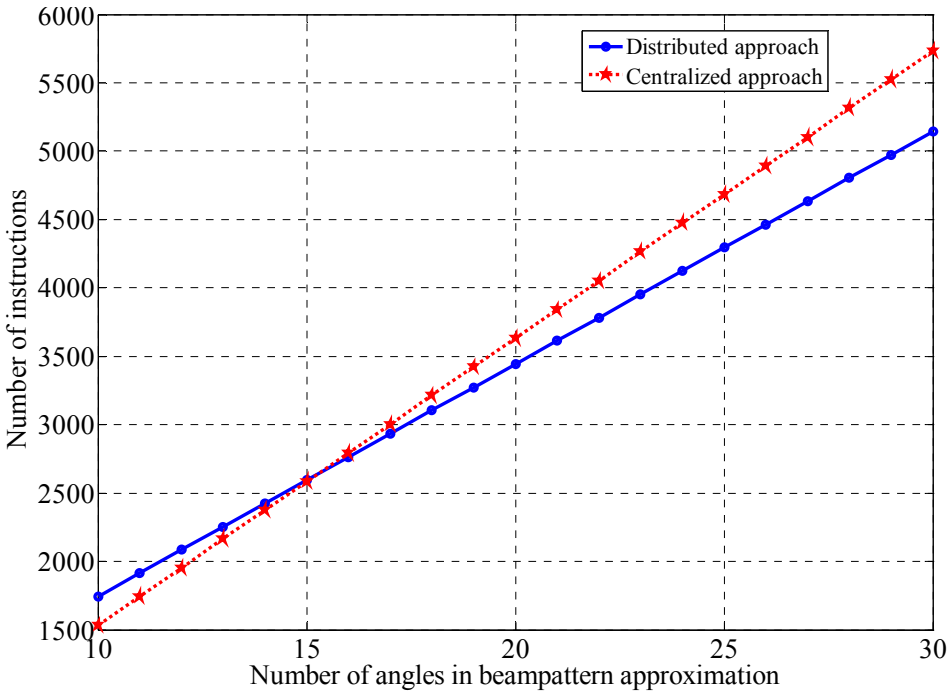


Figure 29. Processing cost of the distributed algorithm as a function of the number of approximation angles for $n = 10$ sensors and $k = 5$ iterations. Multiplying the number of instructions by P_i gives the required processing power.

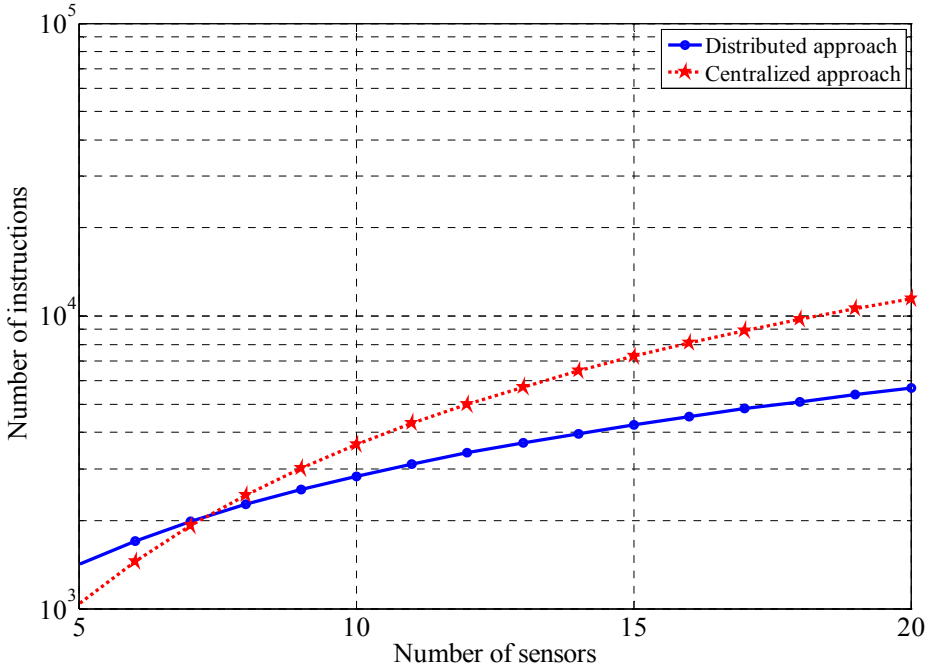


Figure 30. Processing cost of the distributed algorithm as a function of the number of sensors for $m = 20$ approximation angles and $k = 5$ iterations. Multiplying the number of instructions by P_i gives the required processing power.

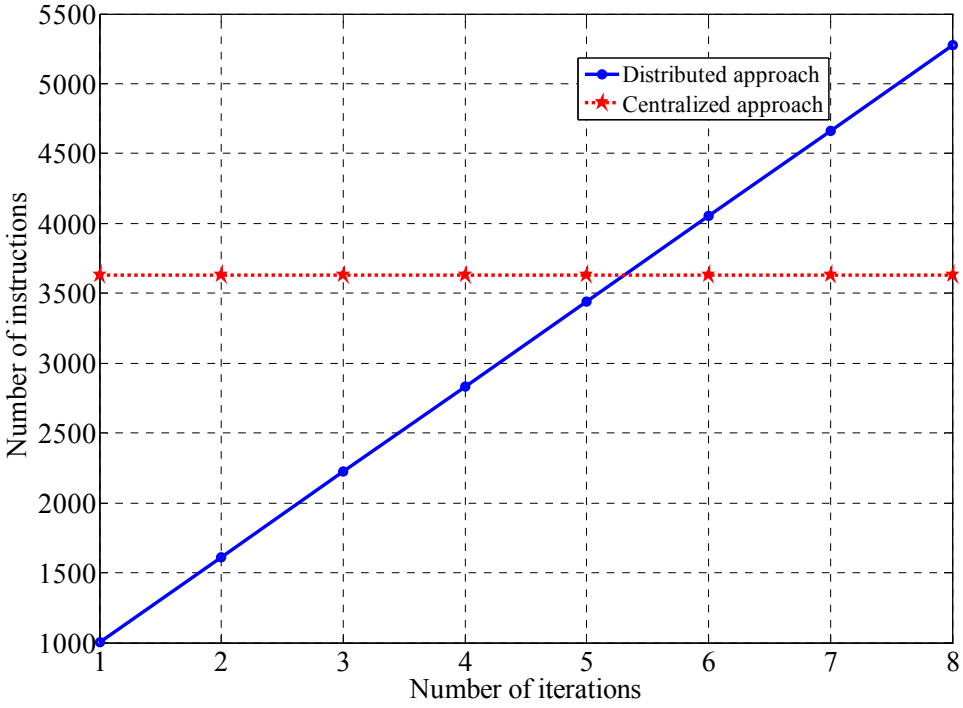


Figure 31. Processing cost of the distributed algorithm as a function of the number of iterations for $n = 10$ sensors and $m = 20$ approximation angles. Multiplying the number of instructions by P_i gives the required processing power.

Finally, Figure 31 shows the increase of the processing cost with the number of iterations, which indicates that the cost for the distributed approach grows higher than the cost for the centralized one after a specific number of iterations ($k = 5$ in this case); a 10×1 sensor array is deployed and twenty approximation points are used.

The communication cost for the distributed approach can be derived as follows. Initially, each node broadcasts its own position, so a total of n data elements are transmitted. Then, after a local iteration step is finished, the scalar correction $s_i^{(k)}$ has to be sent, so for k complete iterations, kn elements are transmitted. Therefore, the number of transmitted elements is

$$C = (k+1)n \tag{100}$$

and it does not depend on the number of approximation angles m. Assuming that each element is represented by b bits, the total transmission power P_c is given by

$$P_c = C \times P_{tb} \times B = (k+1)n \times P_{tb} \times b\,. \tag{101}$$

In Figure 32, the communication cost is plotted as a function of the number of sensors. The increased communication load for the distributed approach is a reasonable tradeoff considering that this distributed approach relieves the central processing node from the entire computational load. Similarly, Figure 33 shows the effect of the number of iterations on the communication cost. Obviously, the speed of convergence positively affects the reduction of the transmitted elements.

The total power for the implementation of the algorithm is the sum of the processing power given by (99) and the transmission power given by (101):

$$P = n\left[2(m-1/3)+k(3m+1)\right] \times P_i + (k+1)n \times P_{tb} \times b \tag{102}$$

Considering the ratio η_{tp} and substituting P_{tb} from (21), the normalized power P_n as a number of required units of power P_i is given by

$$P_n = \frac{P}{P_i} = n\left[2(m-1/3)+k(3m+1)\right] \times P_i + (k+1)n \times \eta_{tp} \times b\,. \tag{103}$$

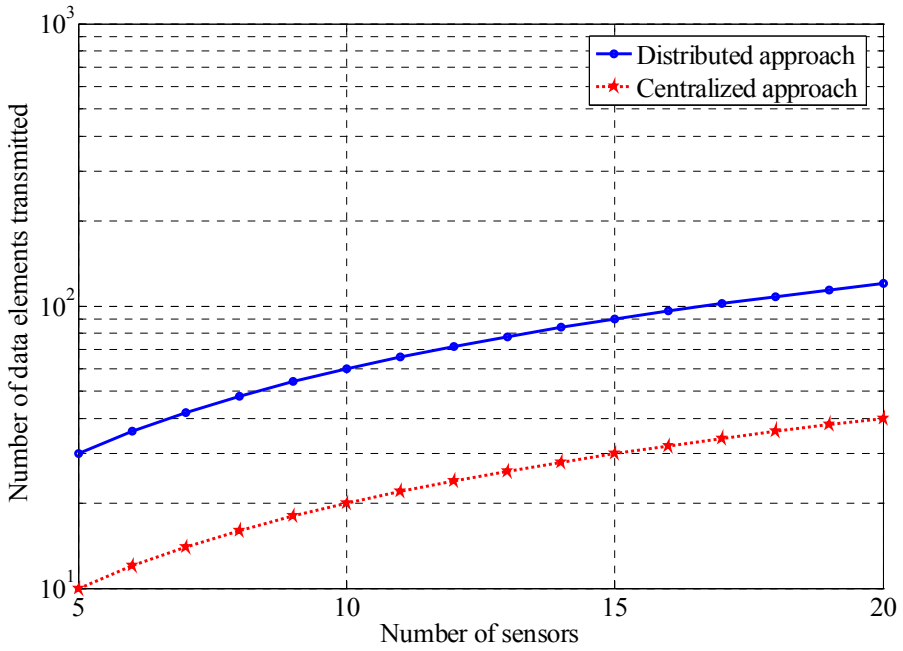


Figure 32. Communication Cost of the distributed algorithm as a function of the number of sensors for $m = 20$ approximation angles and $k = 5$ iterations. Multiplying the number of data elements with $P_{tb} \times b$ gives the required transmission power.

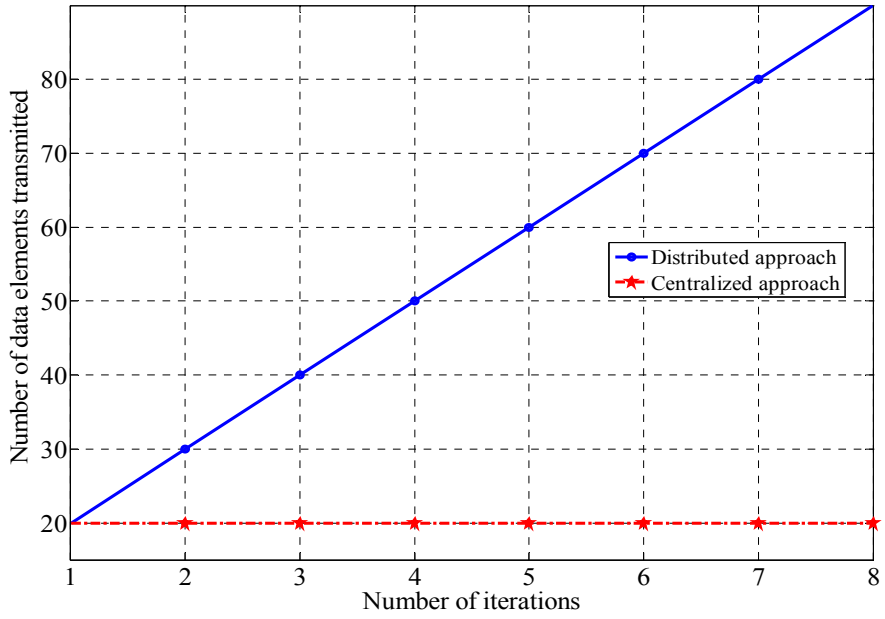


Figure 33. Communication cost of the distributed algorithm as a function of the number of iterations for $n = 10$ sensors and $m = 20$ approximation angles. Multiplying the number of data elements with $P_{tb} \times b$ gives the required transmission power.

In Figures 34-36, the total normalized power needed for the implementation of the algorithm is plotted as a function of the number of approximation angles, the number of sensors and the number of iterations, respectively. In all cases, the power consumption of the distributed algorithm is larger than that of the centralized approach. However, the important characteristic, as in the previous algorithm, is that the total amount of power is shared among the nodes; consequently, the cluster head is relieved of the computational burden. For instance, in Figure 35, where $m = 20$ and $k = 5$, the power for the distributed approach is almost three times larger than the centralized approach, but this is shared by $n = 20$ nodes. Thus, the average power consumption of an arbitrary node in the distributed network is six times less compared to the cluster head's consumption in the centralized architecture. In a centralized approach, there is a considerably higher possibility that the cluster head will exhaust its limited battery and then the beamforming problem will have to be computed from scratch. In summary, the increased power consumption is reasonable enough to consider the distributed approach a viable solution if robustness is required in the network.

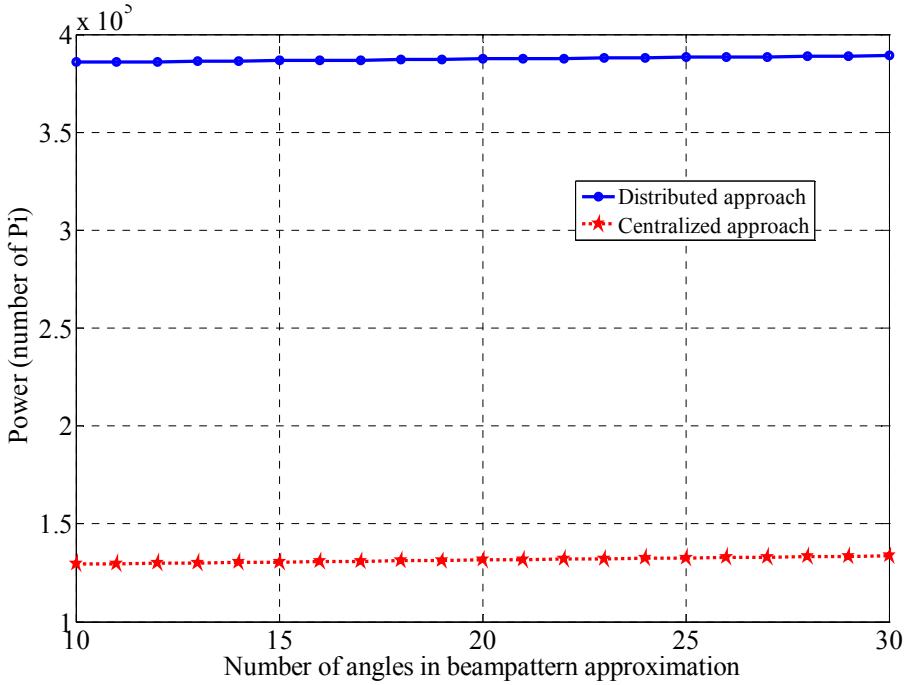


Figure 34. Normalized power P_n (number of P_i) for both distributed and centralized approaches as a function of the number of approximation angles for $n = 10$ sensors and $k = 5$ iterations. η_{tp} is assumed to be 200 and $b = 32$ bits.

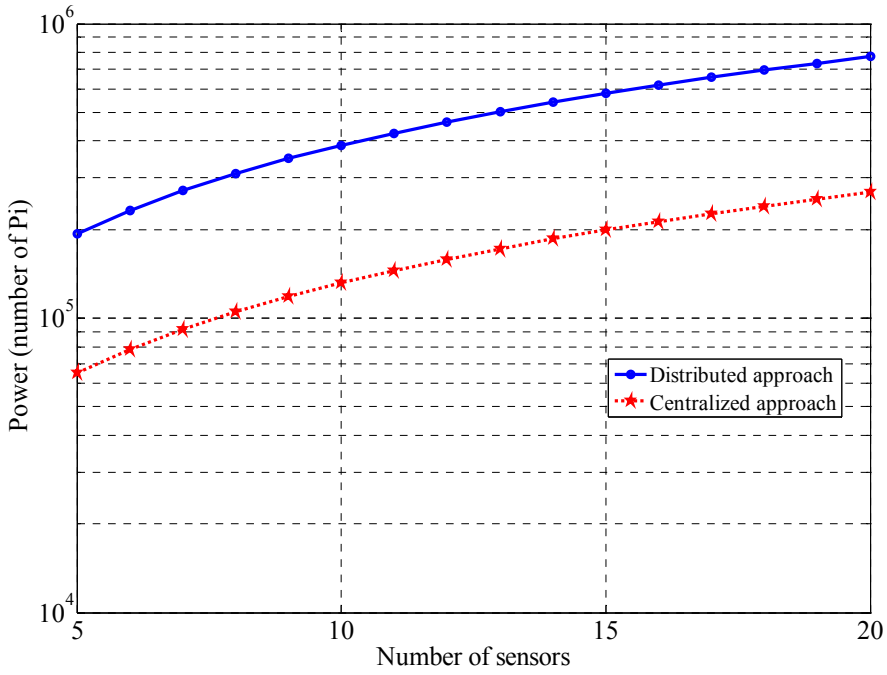


Figure 35. Normalized power P_n (number of P_i) for both distributed and centralized approaches as a function of the number of sensors for $m = 20$ approximation points and $k = 5$ iterations. η_{tp} is assumed to be 200 and $b = 32$ bits.

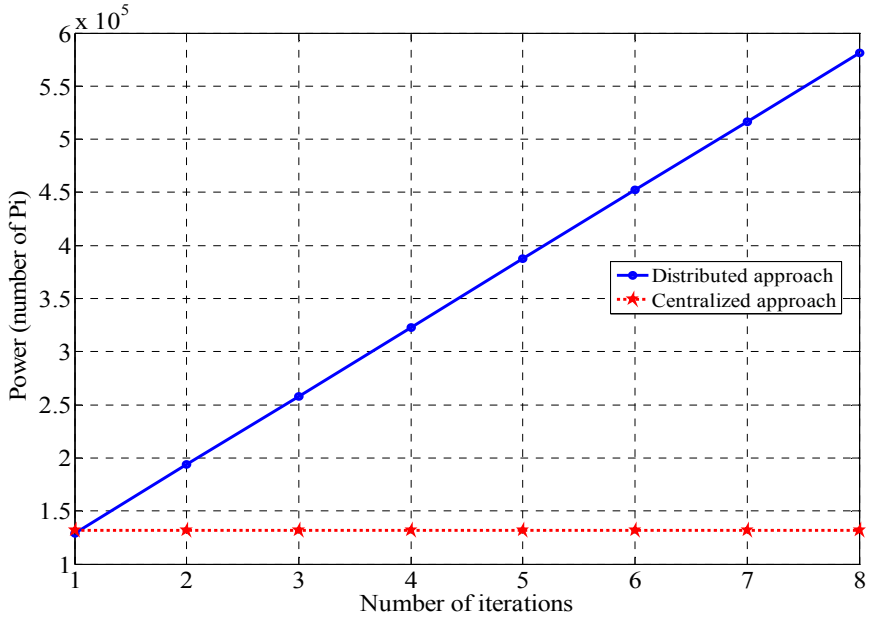


Figure 36. Normalized power P_n (number of P_i) for both distributed and centralized approaches as a function of the number iterations for $n = 10$ sensors and $m = 20$ approximation points. η_{tp} is assumed to be 200 and $b = 32$ bits.

C. SUMMARY

In this chapter, two distributed approaches were proposed for the solution of the LS problem of the array weight computation. The presented algorithms were evaluated in terms of the processing cost, the communication cost and the total power consumption and then compared to the centralized implementation.

In the first approach (distributed QR decomposition), the processing cost is the same as in the centralized approach, but there is a tradeoff of increased communication effort. However, only the first phase of the algorithm is power demanding, and potential modifications to the desired response do not require much power for the recalculation of the weights.

In the second approach (iterative solution), there is rapid convergence to the actual solution, which yields a reduction of the processing cost if the number of iterations is kept low. The simulation results show that only 3 or 4 iterations are needed for the convergence of the algorithm, which results in considerably lower processing power. However, the communication cost is still higher when compared to the centralized approach.

To sum it up, these two approaches require significantly lower average power per sensor node by efficiently sharing the power consumption among the nodes.

V. CONCLUSIONS

The operational scenario adopted in this work assumes that a number of sensor nodes are randomly deployed in an area of interest in order to collect information about various kinds of objects. The acquired data has to be collected by an oveflying UAV; however, single sensors do not have sufficient power capabilities in order to establish communication with the UAV. Therefore, they are organized into clusters and cooperate in order to function as an array of sensor nodes.

The effect of position errors on the performance of the random sensor array was analyzed, and the need for beamforming techniques which effectively mitigate that effect was discussed. Since reliability and robustness in a sensor network environment are crucial, two distributed algorithms for beamforming that efficiently manage to share the processing load among the sensor nodes, compared to centralized approaches which assign the entire effort to a single node, were presented. A simulation model was created and implemented in the MATLAB environment to evaluate the performance of the proposed algorithms.

A. SIGNIFICANT RESULTS

The simulations showed that the sidelobes in the array response increase as a function of the "randomness" of the sensor array. Thus, as the mean deviation from the uniform array was increased, the average sidelobe magnitudes also increased. These results validate the theoretical results of random arrays found in the literature. Another important point is that for the LS beamformer, a subset of approximation points can yield almost the same solution as larger sets. This offers significant reduction to the required processing and transmission power, which are crucial in sensor networks.

Based on the performance analysis, the two proposed distributed algorithms can effectively share the processing load among the nodes. The first, a distributed implementation of the QR decomposition, has the same processing cost as the centralized one. The second approach, based on an iterative method of computing the weight vector

in the LS sense, converges quickly to the actual solution and achieves reduction of the total processing cost compared to that of the centralized one.

For both algorithms, the tradeoff is the increased transmission power, causing an overall increase in the total power consumption in the network. This total power, however, is shared among the sensor nodes; therefore, the average power needed by a sensor node in the distributed implementation is lower than the power needed by the cluster head in the centralized approach. Consequently, the network's susceptibility to failures due to excessive power consumption is greatly reduced.

B. FUTURE WORK

Throughout this work, several assumptions were made, such as the nodes can compute their positions without errors, and the communication between them is not affected by noise. A future effort may examine the effect of these errors on the computation of the weight vector and consequently on the array performance.

In this work, the set of approximation points were selected based on uniform sampling, but there are other choices, such as using a non-uniform grid, which may require fewer points with similar performance. Initial results showed that certain approximation points, which have the physical meaning of direction angles, may be more important for the approximation of the desired response than others. These issues may be further investigated in a future study.

The topology of WSN changes dynamically due to frequent additions and withdrawals of sensor nodes; some of them may switch in or off sleep mode and some other may fail because of the harsh environmental conditions or because of exhausted battery. For these scenarios, the processing and communication cost for the update of the weight vectors can be investigated. Also for the distributed algorithms, there is no need to solve the beamforming problem from scratch; if the array topology changes, it is important to analyze the effects on the costs and consequently the power needed for modifying the weight vector after a node has added or withdrawn from the array.

In this work, the emphasis was given to the data independent beamforming techniques, such as the LS approximation of the desired response. There are many proven data dependent techniques for which the weight vector can be determined adaptively [11], [12] and a distributed implementation of these methods could be examined.

An array of M elements can be used to form a beampattern with exactly $M-1$ narrowband nulls. These nulls can be placed towards the directions of incoming interferences in order to suppress them. Since there are straightforward polynomial based techniques for placing those nulls where desired in the array space, it would be interesting to investigate them in a future work.

Finally, the communication cost was defined as a function of the data elements that need to be transmitted for the implementation of the algorithms. However, there is also a networking cost which consists of parameters such as the packet overhead and the retransmissions due to collisions and errors, which all add to the power consumption. Since the implementation of a distributed beamforming algorithm may be prohibitive by a high networking cost, it will be interesting to investigate the effect of the networking cost to the overall power consumption.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. MATLAB SOURCE CODE

This appendix lists all MATLAB programs used in this work

- **Array2D.m :**

```
%%%     Filename:     Array2D.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This file generates the array beampattern for an
%%%                   array with randomly positioned elements
%%%                   The weights are computed using the LS approach of
%%%                   the desired response


clear all
close all
clc


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

GdBavg=zeros(181,181);
GdBerravg=zeros(181,181);
GdBerrlinavg=zeros(181,181);
GdBrefavg=zeros(181,181);
GdBlsavg=zeros(181,181);
GdBLSavg=zeros(181,181);
GdBiteravg=zeros(181,181,16);

GdBLS1avg=zeros(181,181);
GdBLS2avg=zeros(181,181);
GdBLS3avg=zeros(181,181);
GdBLS4avg=zeros(181,181);
GdBLS5avg=zeros(181,181);

wwls=[];
wwnu=[];
WW1=[];
WW2=[];
WW3=[];
WW4=[];
WW5=[];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  END OF PARAMETERS   %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%   INPUT CHOICES    %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P=inputparameters;

Nx=P(1);          % Nx number of array elements in x direction
Ny=P(2);          % Ny number of array elements in y direction

XG=P(3);          % Generation of position (1):Deviation from perfect
linear, (2): From scratch
xe=P(4);          % Percentage of position error (with respect to
perfect linear) in x direction (%)
  xe=xe/100;
ye=P(5);          % Percentage of position error (with respect to
perfect linear) in y direction (%)
  ye=ye/100;

xest=P(6);        % Percentage of estimated position error (with respect
to actual) in x direction (%)
  xest=xest/100;
yest=P(7);        % Percentage of estimated position error (with respect
to actual) in y direction (%)
  yest=yest/100;

theta0=P(8);      % Elevation angle theta (degrees)
  theta0=theta0*pi/180;
phi0=P(9);        % Azimuth angle  phi (degrees)
  phi0=phi0*pi/180;
phi_ang=P(10);    % Angle phi for beampattern

te=P(11);         % Angle error in theta (+- degrees)
pe=P(12);         %('Angle error in phi (+-degrees)

NumIter=P(13);    %Number of iterations (for average beampattern)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%    END OF INPUT CHOICES   %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%       POSITION GENERATION      %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%    Uniform array (reference)  %%%%%%%%%%

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

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



for NI=1:NumIter;

    %%%%%%%%   DISTANCE DEVIATION  %%%%%%%%%

        if XG==1;
            devx=xe*l*(rand(Nx*Ny,1)-0.5);  % Random deviation from xe%
-lamda/2 to xe% lamda/2
            x=xn+devx;  % Real positions in x-direction

            devy=ye*l*(rand(Nx*Ny,1)-0.5);  % Random deviation from -
lamda/2 to lamda/2
            y=yn+devy;  % Real positions in x-direction

            elseif XG==2;% 2nd option  - Firstly construct x,y and then
assume linear
            [x,y]=rand_inter_dist(Nx,Ny);
        end

 %%%% Move reference node to the axes center %%%%%%%

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


%%%%%%   Estimated position with defined error  %%%%%%%

xerr=x.*(1+xest*(2*rand(Nx*Ny,1)-1));    % error position in x-
direction with respect to actual
yerr=y.*(1+yest*(2*rand(Nx*Ny,1)-1));    % error position in y-
direction with respect to actual

%%%%%%  End of Estimated position with defined error %%%%%%

    %%%%%%%%     END OF DISTANCE DEVIATION     %%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%    WEIGHTS    %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %%%  Amplitudes  %%%

Im=ones(Nx*Ny,1);    % Amplitudes


wm=weights2(x,y,theta0,phi0);           % correct weights
wmerr=weights2(xerr,yerr,theta0,phi0);     % wrong weights, deviation
from actual position : xe%
wmerrlin=weights2(xn,yn,theta0,phi0);    % wrong weights, assume
perfect linear
wref=weights2(xn,yn,theta0,phi0);   % Reference weights (uniform array)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%   END OF WEIGHTS  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%         GAIN          %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


G=gain2D(wm,x,y);          % Gain from correct weights
Gerr=gain2D(wmerr,x,y);    % Gain from wrong weights, deviation from
actual position
Gerrlin=gain2D(wmerrlin,x,y);  % Gain from wrong weights, assumed
perfect linear
Gref=gain2D(wref,xn,yn);


%%%%%%%%  Gain (dB)  %%%%%%%%%%

GdB=10*log10(G/max(max(G)));     % Gain from correct weights (dB)
GdBerr=10*log10(Gerr/max(max(G)));   % Gain from wrong weigths,
deviation from actual position (dB)
GdBerrlin=10*log10(Gerrlin/max(max(G))); % Gain from wrong weights,
assumed perfect linear (dB)
GdBref=10*log10(Gref/max(max(G)));  % Gain for reference (uniform
array)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%   END OF GAIN   %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%%%%%%  LS ESTIMATION OF THE WEIGHTS, GIVEN DESIRED RESPONSE
%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
            theta=-90:90;
            th=theta*pi/180;

            dn=exp(j*b*(xn*sin(th)*cos(phi0)+yn*sin(th)*sin(phi0)));  %
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';

            wwls=[wwls ww];

            Gls=gain2D(ww,x,y);
            GdBls=10*log10(Gls/max(max(G)));



            th0=P(8);

            %%%%%%% LS with fewer approximation points  %%%%

            %% Uniform spacing %%

            %% (1) %%
            dt=2;
            r1a=th0:-dt:-90;
            r1a=flipdim(r1a,2);
            r1b=th0+dt:dt:90;
            r1=[r1a r1b];

            t1=r1*pi/180;

            DN1=exp(j*b*(xn*sin(t1)*cos(phi0)+yn*sin(t1)*sin(phi0)));

            FDES1=wref'*DN1;

            D1=exp(j*b*(x*sin(t1)*cos(phi0)+y*sin(t1)*sin(phi0)));

            ww1=FDES1/D1;
            ww1=ww1';

            WW1=[WW1 ww1];
```

```
GLS1=gain2D(ww1,x,y);
GdBLS1=10*log10(GLS1/max(max(G)));


%% (2) %%
dt=4;
r2a=th0:-dt:-90;
r2a=flipdim(r2a,2);
r2b=th0+dt:dt:90;
r2=[r2a r2b];

t2=r2*pi/180;

DN2=exp(j*b*(xn*sin(t2)*cos(phi0)+yn*sin(t2)*sin(phi0)));

FDES2=wref'*DN2;

D2=exp(j*b*(x*sin(t2)*cos(phi0)+y*sin(t2)*sin(phi0)));

ww2=FDES2/D2;
ww2=ww2';

WW2=[WW2 ww2];

GLS2=gain2D(ww2,x,y);
GdBLS2=10*log10(GLS2/max(max(G)));

%% (3) %%
dt=6;
r3a=th0:-dt:-90;
r3a=flipdim(r3a,2);
r3b=th0+dt:dt:90;
r3=[r3a r3b];

t3=r3*pi/180;

DN3=exp(j*b*(xn*sin(t3)*cos(phi0)+yn*sin(t3)*sin(phi0)));

FDES3=wref'*DN3;

D3=exp(j*b*(x*sin(t3)*cos(phi0)+y*sin(t3)*sin(phi0)));

ww3=FDES3/D3;
ww3=ww3';

WW3=[WW3 ww3];

GLS3=gain2D(ww3,x,y);
GdBLS3=10*log10(GLS3/max(max(G)));
```

```
%% (4) %%
dt=8;
r4a=th0:-dt:-90;
r4a=flipdim(r4a,2);
r4b=th0+dt:dt:90;
r4=[r4a r4b];

t4=r4*pi/180;

DN4=exp(j*b*(xn*sin(t4)*cos(phi0)+yn*sin(t4)*sin(phi0)));

FDES4=wref'*DN4;

D4=exp(j*b*(x*sin(t4)*cos(phi0)+y*sin(t4)*sin(phi0)));

ww4=FDES4/D4;
ww4=ww4';

WW4=[WW4 ww4];

GLS4=gain2D(ww4,x,y);
GdBLS4=10*log10(GLS4/max(max(G)));


%% (5) %%
dt=10;
r5a=th0:-dt:-90;
r5a=flipdim(r5a,2);
r5b=th0+dt:dt:90;
r5=[r5a r5b];

t5=r5*pi/180;

DN5=exp(j*b*(xn*sin(t5)*cos(phi0)+yn*sin(t5)*sin(phi0)));

FDES5=wref'*DN5;

D5=exp(j*b*(x*sin(t5)*cos(phi0)+y*sin(t5)*sin(phi0)));

ww5=FDES5/D5;
ww5=ww5';

WW5=[WW5 ww5];

GLS5=gain2D(ww5,x,y);
GdBLS5=10*log10(GLS5/max(max(G)));


%%% Non uniform %%%%
theta1=th0-12:4:th0+12;
theta2=-90:15:90;
```

```
            ft=(theta2<th0-12)|(theta2>th0+12);
            thetaf=[theta1 theta2(ft)];
            THETA=sort(thetaf);

            TH=THETA*pi/180;

            DN=exp(j*b*(xn*sin(TH)*cos(phi0)+yn*sin(TH)*sin(phi0)));

            FDES=wref'*DN;

            D=exp(j*b*(x*sin(TH)*cos(phi0)+y*sin(TH)*sin(phi0)));

            WW=FDES/D;
            WW=WW';

            wwnu=[wwnu WW];

            GLS=gain2D(WW,x,y);
            GdBLS=10*log10(GLS/max(max(G)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%     END OF LS         %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%    PLOTS    %%%%%%%%%%%%%

figure(10);

plot(xn,yn,'o')
hold on;
plot(x,y,'rx')
hold on;
plot(xerr,yerr,'mp')
grid on
axis equal
title('Fig.1 Sensor array','Fontsize',12);
legend('Perfect linear','Actual Position','Wrongly estimated');


figure(20);
plot(theta,GdB(:,phi_ang+90+1),'Linewidth',2);
hold on;
plot(theta,GdBerr(:,phi_ang+90+1),'r-.','Linewidth',2);
hold on;
plot(theta,GdBerrlin(:,phi_ang+90+1),'m:','Linewidth',2);
hold on;
plot(theta,GdBref(:,phi_ang+90+1),'g-','Linewidth',2);
hold on;
plot(theta,GdBls(:,phi_ang+90+1),'c-','Linewidth',2);
axis([-85 85 -50 5]);
grid on
title('Fig.2 : Beampattern for N linear array elements and given
\phi','Fontsize',12);
```

```
xlabel('\theta (degrees)','Fontsize',12);
ylabel('Power Gain (dB),' ,'Fontsize',12);
legend('Correct','Wrongly estimated','Assumed perfect linear','Ideal
linear','LS weights');


GdBavg=GdBavg+GdB;
GdBerravg=GdBerravg+GdBerr;
GdBerrlinavg=GdBerrlinavg+GdBerrlin;
GdBrefavg=GdBrefavg+GdBref;
GdBlsavg=GdBlsavg+GdBls;
GdBLSavg=GdBLSavg+GdBLS;

GdBLS1avg=GdBLS1avg+GdBLS1;
GdBLS2avg=GdBLS2avg+GdBLS2;
GdBLS3avg=GdBLS3avg+GdBLS3;
GdBLS4avg=GdBLS4avg+GdBLS4;
GdBLS5avg=GdBLS5avg+GdBLS5;

end     %%%%% End of loop for iterated computations (average Gain)
%%%%%%%

GdBavg=GdBavg/NumIter;
GdBerravg=GdBerravg/NumIter;
GdBerrlinavg=GdBerrlinavg/NumIter;
GdBrefavg=GdBrefavg/NumIter;
GdBlsavg=GdBlsavg/NumIter;
GdBLSavg=GdBLSavg/NumIter;

GdBLS1avg=GdBLS1avg/NumIter;
GdBLS2avg=GdBLS2avg/NumIter;
GdBLS3avg=GdBLS3avg/NumIter;
GdBLS4avg=GdBLS4avg/NumIter;
GdBLS5avg=GdBLS5avg/NumIter;

figure(30);
theta=-90:90;
plot(theta,GdBavg(:,phi_ang+90+1),'Linewidth',2);
hold on;
plot(theta,GdBerravg(:,phi_ang+90+1),'r-.','Linewidth',2);
hold on;
plot(theta,GdBerrlinavg(:,phi_ang+90+1),'m:','Linewidth',2);
hold on;
plot(theta,GdBrefavg(:,phi_ang+90+1),'g-','Linewidth',2);
hold on;
plot(theta,GdBlsavg(:,phi_ang+90+1),'c-','Linewidth',2);
grid on
legend('Correct','Wrongly estimated','Assumed perfect linear','Ideal
linear','LS weights');
axis([-85 85 -50 5]);
title('Fig.3 : Average Beampattern for N linear array elements and
given \phi','Fontsize',12);
xlabel('\theta (degrees)','Fontsize',12);
ylabel('Power Gain (dB),' ,'Fontsize',12);
```

```
figure(40);
theta=-90:90;
plot(theta,GdBavg(:,phi_ang+90+1),'Linewidth',2);
hold on;
plot(theta,GdBerravg(:,phi_ang+90+1),'r-.','Linewidth',2);
hold on;
plot(theta,GdBrefavg(:,phi_ang+90+1),'g-','Linewidth',2);
hold on;
plot(theta,GdBlsavg(:,phi_ang+90+1),'c-','Linewidth',2);
axis([-85 85 -50 5]);
title('Fig.4 : Average Beampattern for N linear array
elements','Fontsize',12);
xlabel('\theta (degrees)','Fontsize',12);
ylabel('Power Gain (dB),' ,'Fontsize',12);
grid on
legend('Correct','Wrongly estimated','Ideal linear','LS weights');

figure(50);
theta=-90:90;
plot(theta,GdBavg(:,phi_ang+90+1),'Linewidth',2);
hold on;
plot(theta,GdBrefavg(:,phi_ang+90+1),'g-','Linewidth',2);
hold on;
plot(theta,GdBlsavg(:,phi_ang+90+1),'c-','Linewidth',2);
hold on;
plot(theta,GdBLSavg(:,phi_ang+90+1),'k:','Linewidth',2);
axis([-85 85 -50 5]);
title('Fig.5 : Average Beampattern for N linear array
elements','Fontsize',12);
xlabel('\theta (degrees)','Fontsize',12);
ylabel('Power Gain (dB),' ,'Fontsize',12);
grid on
legend('Correct','Ideal linear','LS weights','LS less constraints');
```

- **Array2D.m :**

```
%%%     Filename:     InputParameters.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This function creates a GUI for defining the
%%%                   characteristics of a random array

function answer=inputparameters;


prompt={'Give Nx number of array elements in x direction:',...
        'Give Ny number of array elements in y direction:',...
        'Generation of position: (1):Deviation from perfect linear,(2):
From scratch',...
        'Position error (with respect to perfect linear) in x direction
(%)',...
        'Position error (with respect to perfect linear) in y direction
(%)',...
        'Estimated position error (with respect to actual) in x
direction (%)',...
        'Estimated position error (with respect to actual) in y
direction (%)',...
        'Elevation angle (theta) (degrees):',...
        'Azimuth angle (phi) (degrees):',...
        'Angle phi for beampattern:',...
        'Angle error in theta (+- degrees)',...
        'Angle error in phi (+-degrees)',...
        'Number of iterations',};

name='Parameters for antenna array';
numlines=1;
defaultanswer={'10','1','1','20','20','0','0','30','45','45','0','0','2
5'};

answer=inputdlg(prompt,name,numlines,defaultanswer);

for i=1:length(answer);
    temp(i)=str2num(answer{i});
end

answer=temp;
```

- **Gain2D.m :**

```
%%%     Filename:     Gain2D.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This function calculates the beampattern gain of
%%%                   an array


function Gain=gain2(wm,xm,ym);

global b

for theta=-90:90;
   for phi=-90:90;
        th=theta*pi/180;
        ph=phi*pi/180;

F(90+theta+1,90+phi+1)=sum(sum(conj(wm).*exp(j*b*(xm*sin(th)*cos(ph)+ym
*sin(th)*sin(ph)))));
   end
end

Gain=abs(F).^2;
```

- **weights2.m :**

```
%%%     Filename:     weights2.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This function calculates the weights for a
uniform
%%%                   array


function w=weights2(xm,ym,Theta,Phi);


global Im b

w=Im.*exp(j*b*(xm*sin(Theta)*cos(Phi)+ym*sin(Theta)*sin(Phi)));  %
weights
```

- **rand_inter_dist.m :**

```
%%%     Filename:     rand_inter_dist.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This function creates a random array. the
deviations
%%%                   from the ideal array follow a uniform
distribution


function [xx,yy]=rand_inter_dist(Nx,Ny);

global l

Ny=Ny+1;    % First line will be ignored
Nx=Nx+1;

xx=zeros(Ny,Nx);
yy=zeros(Ny,Nx);

for i=1:Ny;
    for j=1:Nx;
        if j==1
            xx(i,j)=0;
        else
            xx(i,j)=xx(i,j-1)+(rand*l/2+l/4);
        end

        if i==1;
            yy(i,j)=0;
        else
            yy(i,j)=yy(i-1,j)+(rand*l/2+l/4);
        end
    end
end

if Ny~=1
    xx1=xx(2:Ny,2:Nx);
    yy1=yy(2:Ny,2:Nx);
end

xx1=reshape(xx1,(Nx-1)*(Ny-1),1);   % ignore first line
yy1=reshape(yy1,(Nx-1)*(Ny-1),1);

xx=xx1;
yy=yy1;
```

- **CostAnalysis.m :**

```
%%%     Filename:     CostAnalysis.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This file is used for the calculation of the
%%%                   processing and communication costs and the
%%%                   power consumption for the
%%%                   centralized,the distributed QR decomposition
%%%                   and the iterative approach.

clear all
close all
clc

N=10;    % Number of sensors
M=10:30;   % Number of angles

%% Reference
%% Computational Cost
Comp_ref=2*N^2*(M-N/3)+M*N+N^2;
Com_ref1=2*N;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%  Distributed QR decomposition %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%  COMPUTATIONAL COST
%%  Number of Operations)
%%  Note that there is no distinguish between the type of operations
%%  like additions or multiplications

%   A matrix (M x N)
%   b vector (M x 1)
%   QR decomposition :  Cqr=2*N^2*(M-N/3)
%   Update           :  Cu=M*N
%   Back Substitution:  Cb=N^2


Cqr=2*N^2*(M-N/3);
Cu=M*N;
Cb=N^2;
Cub=Cu+Cb;

Comp1=Cqr+Cub;

figure(1);
semilogy(M,Comp1,'bo-',M,Cqr,'rp-.',M,Cub,'md--','Linewidth',1.5);
```

```
title('Fig.1 : Computational Cost as a function of M ,
(N=10)','Fontsize',12);
xlabel('Number of angles in beampattern approximation','Fontsize',12);
ylabel('Number of instructions','Fontsize',12);
legend('Total','QR decomposition','Update and Back Substitution');
grid on;


M=20;
N=5:20;
Cqr=2*N.^2.*(M-N/3);
Cu=M*N;
Cb=N.^2;
Cub=Cu+Cb;

Comp2=Cqr+Cub;

figure(2);
semilogy(N,Comp2,'bo-',N,Cqr,'rp-.',N,Cub,'md--','Linewidth',1.5);
title('Fig.1 : Computational Cost as a function of M ,
(N=10)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Number of instructions','Fontsize',12);
legend('Total','QR decomposition','Update and Back Substitution');
grid on;


%% COMMUNICATION COST
%% Defined as the number of data values we need to send (broadcasting)
%% Not the number of packets

% 1st pass : From 1st sensor to the last (QR decomposition) :
%           Cqr=(M+4-N/2)*(N-1)
% 2nd pass : Back substitution phase
%           Cbs=2*(N-1)

N=10;    % Number of sensors
M=10:30;   % Number of angles

Comm_qr=(M+2-N/2)*(N-1);
Comm_bs=2*(N-1);

Com1=Comm_qr+Comm_bs;

figure(10);
semilogy(M,Com1,'bo-',M,Comm_qr,'rp-
.',M,ones(length(M),1)*Comm_bs,'md:',M,ones(length(M),1)*Com_ref1,'gs-
.','Linewidth',1.5);
title('Fig.3 :Communication Cost as a function of M ,
(N=10)','Fontsize',12);
xlabel('Number of angles in beampattern approximation','Fontsize',12);
ylabel('Number of elements transmitted','Fontsize',12);
```

```
legend('Total','QR decomposition ','Update and Back
Substitution','Centralized approach');
grid on;


M=20;
N=5:20;
Com_ref2=2*N;

Comm_qr=(M+2-N/2).*(N-1);
Comm_bs=2*(N-1);

Com2=Comm_qr+Comm_bs;


figure(11);
semilogy(N,Com2,'bo-',N,Comm_qr,'rp-.',N,Comm_bs,'md:',N,Com_ref2,'gs-
.','Linewidth',1.5);
title('Fig.4 :Communication Cost as a function of N ,
(M=20)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Number of elements transmitted','Fontsize',12);
legend('Total','QR decomposition ','Update and Back
Substitution','Centralized approach');
grid on;


%%% Power analysis
Ntp=200;
Pi=1;
Ptb=Ntp*Pi;
B=32;

% total Power

% vs M
P1=Comp1*Pi+Com1*B*Ptb;
Pref1=Comp1*Pi+Com_ref1*B*Ptb;

N=10;    % Number of sensors
M=10:30;   % Number of angles

figure(20);
semilogy(M,P1,'bo-',M,Pref1,'rp:','Linewidth',1.5);
title('Fig.20 :Power analysis as a function of M ,
(N=10)','Fontsize',12);
xlabel('Number of angles in beampattern approximation','Fontsize',12);
ylabel('Power (number of Pi)','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;

% vs N
P2=Comp2*Pi+Com2*B*Ptb;
```

```
Pref2=Comp2*Pi+Com_ref2*B*Ptb;

M=20;
N=5:20;

figure(25);
semilogy(N,P2,'bo-',N,Pref2,'rp:','Linewidth',1.5);
title('Fig.25 :Power analysis as a function of N ,
(M=20)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Power (number of Pi)','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%  Iterative Approach  %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


N=10;    % Number of sensors
M=10:30;   % Number of angles
K=5;

CP1=N*(2*(M-1/3)+K*(3*M+1));

figure(30);
semilogy(M,CP1,'bo-',M,Comp1,'rp:','Linewidth',1.5);
title('Fig.30 :Computational Cost as a function of M ,
(N=10,K=5)','Fontsize',12);
xlabel('Number of angles in beampattern approximation','Fontsize',12);
ylabel('Number of instructions','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;



M=20;
N=5:20;
K=4;

CP2=N*(2*(M-1/3)+K*(3*M+1));

figure(35);
semilogy(N,CP2,'bo-',N,Comp2,'rp:','Linewidth',1.5);
title('Fig.35 :Computational Cost as a function of N ,
(M=20)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Number of instructions','Fontsize',12);
legend('Distributed approach','Centralized approach');
```

```
grid on;


M=20;
N=10;
K=1:8;

CP3=N*(2*(M-1/3)+K*(3*M+1));
Cref=2*N^2*(M-N/3)+M*N+N^2;

figure(40);
semilogy(K,CP3,'bo-',K,Cref*ones(length(K),1),'rp:','Linewidth',1.5);
title('Fig.40 :Computational Cost as a function of iterations K ,
(N=10,M=20)','Fontsize',12);
xlabel('Number of iterations','Fontsize',12);
ylabel('Number of instructions','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;



N=10;    % Number of sensors
M=10:30;   % Number of angles
K=5;
ComPar1=(K+1)*N;

figure(50)

M=20;
N=5:20;
K=5;

ComPar2=(K+1)*N;

semilogy(N,ComPar2,'bo-',N,Com_ref2,'rp:','Linewidth',1.5);
title('Fig.50 :Communication Cost as a function of N ,
(M=20,K=5)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Number of elements transmitted','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;


figure(60);

M=20;
N=10;
K=1:8;

ComPar3=(K+1)*N;
Com_ref3=2*N;
```

```
semilogy(K,ComPar3,'bo-
',K,ones(length(K),1)*Com_ref3,'rp:','Linewidth',1.5);
title('Fig.60 :Communication Cost as a function of K ,
(N=10,M=20)','Fontsize',12);
xlabel('Number of iterations','Fontsize',12);
ylabel('Number of elements transmitted','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;


%%% Power Analysis
Ntp=200;
Pi=1;
Ptb=Ntp*Pi;
B=32;


N=10;    % Number of sensors
M=10:30;   % Number of angles
K=4;

Par1=CP1*Pi+ComPar1*B*Ptb;
Pref1=Comp1*Pi+Com_ref1*B*Ptb;

figure(70);
plot(M,Par1,'bo-',M,Pref1,'rp:','Linewidth',1.5);
title('Fig.70 :Power analysis as a function of M ,
(N=10)','Fontsize',12);
xlabel('Number of angles in beampattern approximation','Fontsize',12);
ylabel('Power (number of Pi)','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;


M=20;
N=5:20;
K=4;

Par2=CP2*Pi+ComPar2*B*Ptb;
Pref2=Comp2*Pi+Com_ref2*B*Ptb;

figure(80);
semilogy(N,Par2,'bo-',N,Pref2,'rp:','Linewidth',1.5);
title('Fig.80 :Power analysis as a function of N ,
(M=20)','Fontsize',12);
xlabel('Number of sensors','Fontsize',12);
ylabel('Power (number of Pi)','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;


M=20;
N=10;
```

```
K=1:8;

Par3=CP3*Pi+ComPar3*B*Ptb;

Cqr=2*N.^2.*(M-N/3);
Cu=M*N;
Cb=N.^2;
Cub=Cu+Cb;

Comp3=Cqr+Cub;

Pref3=Comp3*Pi+Com_ref3*B*Ptb;

figure(90);
plot(K,Par3,'bo-',K,ones(length(K),1)*Pref3,'rp:','Linewidth',1.5);
title('Fig.90 :Power analysis as a function of K ,
(M=20)','Fontsize',12);
xlabel('Number of iterations','Fontsize',12);
ylabel('Power (number of Pi)','Fontsize',12);
legend('Distributed approach','Centralized approach');
grid on;
```

- **Iterative.m :**

```
%%%     Filename:     Iterative.m
%%%     Author:       Nikolaos Papalexidis
%%%                   Hellenic Air Force
%%%     Date:         June 2007
%%%     Description:  This file is used for the implementation of the
%%%                   distributed iterative solution of the LS problem


xpos=x;
ypos=y;

A=D5';
q=FDES5';
[M,N]=size(A);

clear z1 z2 z3 z4 z5 z6 z7 z8 zz n e

Z1=[];
Z2=[];
Z3=[];
Z4=[];
Z5=[];
Z6=[];
Z7=[];
Z8=[];

z1=0;
z2=0;
z3=0;
z4=0;
z5=0;
z6=0;
z7=0;
z8=0;

A1=A(:,1);
A2=A(:,2);
A3=A(:,3);
A4=A(:,4);
A5=A(:,5);
A6=A(:,6);
A7=A(:,7);
A8=A(:,8);

zz1=zeros(N,1);
zz1(1)=z1;
zz2=zeros(N,1);
zz2(2)=z2;
zz3=zeros(N,1);
zz3(3)=z3;
zz4=zeros(N,1);
```

```
zz4(4)=z4;
zz5=zeros(N,1);
zz5(5)=z5;
zz6=zeros(N,1);
zz6(6)=z6;
zz7=zeros(N,1);
zz7(7)=z7;
zz8=zeros(N,1);
zz8(8)=z8;

s1=0;
s2=0;
s3=0;
s4=0;
s5=0;
s6=0;
s7=0;
s8=0;

zz=zz1+zz2+zz3+zz4+zz5+zz6+zz7+zz8;

r0=A*zz(:,1)-q;
Niter=6;   % Number of iterations
r=zeros(M,Niter*N);   % residual
r(:,1)=r0;
rhat=r0;
z=A\q;

for k=1:Niter;
    m=(k-1)*8+1;

    s1=-A1\rhat;
    rhat=rhat+A1*s1;
    r(:,m+1)=rhat;
    z1=z1+s1;
    Z1=[Z1 z1];

    s2=-A2\rhat;
    rhat=rhat+A2*s2;
    r(:,m+2)=rhat;
    z2=z2+s2;
    Z2=[Z2 z2];

    s3=-A3\rhat;
    rhat=rhat+A3*s3;
    r(:,m+3)=rhat;
    z3=z3+s3;
    Z3=[Z3 z3];

    s4=-A4\rhat;
    rhat=rhat+A4*s4;
    r(:,m+4)=rhat;
    z4=z4+s4;
    Z4=[Z4 z4];
```

```
    s5=-A5\rhat;
    rhat=rhat+A5*s5;
    r(:,m+5)=rhat;
    z5=z5+s5;
    Z5=[Z5 z5];;

    s6=-A6\rhat;
    rhat=rhat+A6*s6;
    r(:,m+6)=rhat;
    z6=z6+s6;
    Z6=[Z6 z6];

    s7=-A7\rhat;
    rhat=rhat+A7*s7;
    r(:,m+7)=rhat;
    z7=z7+s7;
    Z7=[Z7 z7];

    s8=-A8\rhat;
    rhat=rhat+A8*s8;
    r(:,m+8)=rhat;
    z8=z8+s8;
    Z8=[Z8 z8];

    zz(:,k)=[z1;z2;z3;z4;z5;z6;z7;z8];
end

for i=1:Niter*N+1;
    n(i)=norm(r(:,i));
end

R=norm(A*z-q);

figure(1);
t1=1:length(Z1);
N1=length(t1);

subplot(3,3,1);
plot(t1,ones(N1,1)*real(z(1)),'r-',t1,real(Z1),'b-.');
title('Real of w_1','Fontsize',12);
grid on;

subplot(3,3,2);
plot(t1,ones(N1,1)*real(z(2)),'r-',t1,real(Z2),'b-.');
title('Real of w_2','Fontsize',12);
grid on;

subplot(3,3,3);
plot(t1,ones(N1,1)*real(z(3)),'r-',t1,real(Z3),'b-.');
title('Real of w_3','Fontsize',12);
grid on;
```

```
subplot(3,3,4);
plot(t1,ones(N1,1)*real(z(4)),'r-',t1,real(Z4),'b-.');
title('Real of w_4','Fontsize',12);
grid on;

subplot(3,3,5);
plot(t1,ones(N1,1)*real(z(5)),'r-',t1,real(Z5),'b-.');
title('Real of w_5','Fontsize',12);
grid on;

subplot(3,3,6);
plot(t1,ones(N1,1)*real(z(6)),'r-',t1,real(Z6),'b-.');
title('Real of w_6','Fontsize',12);
grid on;

subplot(3,3,7);
plot(t1,ones(N1,1)*real(z(7)),'r-',t1,real(Z7),'b-.');
title('Real of w_7','Fontsize',12);
grid on;

subplot(3,3,8);
plot(t1,ones(N1,1)*real(z(8)),'r-',t1,real(Z8),'b-.');
title('Real of w_8','Fontsize',12);
grid on;

subplot(3,3,9);
plot(n);
hold on;
plot(ones(Niter*N+1,1)*R,'r-');
title('Residual Norm','Fontsize',12);
grid on;



figure(2);
t1=1:length(Z1);
N1=length(t1);

subplot(3,3,1);
plot(t1,ones(N1,1)*imag(z(1)),'r-',t1,imag(Z1),'b-.');
title('Imaginary of w_1','Fontsize',12);
grid on;

subplot(3,3,2);
plot(t1,ones(N1,1)*imag(z(2)),'r-',t1,imag(Z2),'b-.');
title('Imaginary of w_2','Fontsize',12);
grid on;

subplot(3,3,3);
plot(t1,ones(N1,1)*imag(z(3)),'r-',t1,imag(Z3),'b-.');
title('Imaginary of w_3','Fontsize',12);
grid on;
```

```
subplot(3,3,4);
plot(t1,ones(N1,1)*imag(z(4)),'r-',t1,imag(Z4),'b-.');
title('Imaginary of w_4','Fontsize',12);
grid on;

subplot(3,3,5);
plot(t1,ones(N1,1)*imag(z(5)),'r-',t1,imag(Z5),'b-.');
title('Imaginary of w_5','Fontsize',12);
grid on;

subplot(3,3,6);
plot(t1,ones(N1,1)*imag(z(6)),'r-',t1,imag(Z6),'b-.');
title('Imaginary of w_6','Fontsize',12);
grid on;

subplot(3,3,7);
plot(t1,ones(N1,1)*imag(z(7)),'r-',t1,imag(Z7),'b-.');
title('Imaginary of w_7','Fontsize',12);
grid on;

subplot(3,3,8);
plot(t1,ones(N1,1)*imag(z(8)),'r-',t1,imag(Z8),'b-.');
title('Imaginary of w_8','Fontsize',12);
grid on;

subplot(3,3,9);
plot(n);
hold on;
plot(ones(Niter*N+1,1)*R,'r-');
title('Residual Norm','Fontsize',12);
grid on;



figure(3);
t1=1:length(Z1);
N1=length(t1);

subplot(3,3,1);
plot(t1,ones(N1,1)*abs(z(1)),'r-',t1,abs(Z1),'b-.');
title('Magnitude of w_1','Fontsize',12);
grid on;

subplot(3,3,2);
plot(t1,ones(N1,1)*abs(z(2)),'r-',t1,abs(Z2),'b-.');
title('Magnitude of w_2','Fontsize',12);
grid on;

subplot(3,3,3);
plot(t1,ones(N1,1)*abs(z(3)),'r-',t1,abs(Z3),'b-.');
title('Magnitude of w_3','Fontsize',12);
grid on;
```

```
subplot(3,3,4);
plot(t1,ones(N1,1)*abs(z(4)),'r-',t1,abs(Z4),'b-.');
title('Magnitude of w_4','Fontsize',12);
grid on;

subplot(3,3,5);
plot(t1,ones(N1,1)*abs(z(5)),'r-',t1,abs(Z5),'b-.');
title('Magnitude of w_5','Fontsize',12);
grid on;

subplot(3,3,6);
plot(t1,ones(N1,1)*abs(z(6)),'r-',t1,abs(Z6),'b-.');
title('Magnitude of w_6','Fontsize',12);
grid on;

subplot(3,3,7);
plot(t1,ones(N1,1)*abs(z(7)),'r-',t1,abs(Z7),'b-.');
title('Magnitude of w_7','Fontsize',12);
grid on;

subplot(3,3,8);
plot(t1,ones(N1,1)*abs(z(8)),'r-',t1,abs(Z8),'b-.');
title('Magnitude of w_8','Fontsize',12);
grid on;

subplot(3,3,9);
plot(n);
hold on;
plot(ones(Niter*N+1,1)*R,'r-');
title('Residual Norm','Fontsize',12);
grid on;


figure(4);
semilogy(n,'bo-');
hold on;
semilogy(ones(Niter*N+1,1)*R,'rp-');
title('Residual Norm','Fontsize',12);
xlabel('Number of local iterations','Fontsize',12);
ylabel('Residual Norm','Fontsize',12);
grid on;


figure(5);

for i=1:Niter;
    ztemp=[Z1(i);Z2(i);Z3(i);Z4(i);Z5(i);Z6(i);Z7(i);Z8(i)];
    e(i)=norm(z-ztemp);
end

figure(6)
semilogy(e,'bo-');
title('Norm of the weight error','Fontsize',12);
```

```
xlabel('Number of complete iterations','Fontsize',12);
ylabel('Norm of the error between approximate and actual
weights','Fontsize',12);
grid on;
```

LIST OF REFERENCES

[1] I.F. Akyildiz, et al., "A Survey on Sensor Networks," *IEEE Communications Magazine*, Vol.40, No.8, pp. 102-114, August 2002.

[2] D.E. Culler and W. Hong, "Wireless Sensor Networks," *Communications of the ACM*, Vol.47, No.6, pp. 30-33, June 2004.

[3] Q. Zhao, A. Swami and L. Tong, "The Interplay Between Signal Processing and Networking in Sensor Networks," *IEEE Signal Processing Magazine*, Vol.23, No.4, pp. 84-93, July 2006.

[4] J.J. Xiao, et al., "Distributed Compression-Estimation Using Wireless Sensor Networks," *IEEE Signal Processing Magazine*, Vol.23, No.4, pp. 27-41, July 2006.

[5] I. Stojmenovic, "*Handbook of Sensor Networks: Algorithms and Architectures* ," John Wiley & Sons, Hoboken, NJ, 2005.

[6] C.S. Raghavendra, Krishna M. Sivalingam and T. Znati, "*Wireless Sensor Networks,*" Kluwer Academic Publishers, Norwell, MA, 2004.

[7] D.E. Culler, D. Estrin and M. Srivastava, "Overview of Sensor Networks," *Computer*, Vol.37, No.8, pp. 41-49, August 2004.

[8] L.C. Godara, "Application of Antenna Arrays to Mobile Communications, Part II: Beam-forming and Direction-of-Arrival Considerations," *Proceedings of the IEEE*, Vol.85, No.8, pp. 1195-1245, August 1997.

[9] W. L. Stutzman and G.A. Thiele., *Antenna Theory and Design*, John Wiley & Sons, New York, 1998.

[10] L.C. Godara, "*Handbook of antennas in Wireless Communications*," CRC Press, Boca Raton, FL 33431, 2002.

[11] B.D. Van Veen, and K.M. Buckley, "Beamforming: A Versatile Approach to Spatial Filtering", *IEEE ASSP Magazine*, pp. 4-24, April 1988.

[12] S. Haykin, "*Adaptive Filter Theory*," Prentice Hall, Upper Saddle River, NJ, 2002.

[13] T. Biedka, "*Analysis and Development of Blind Adaptive Algorithms*," Ph.D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA, October 2001.

[14] D.G. Manolakis, Ingle V.K. and Kogon, S.M., "*Statistical and Adaptive Signal Processing*," Artech House, Norwood, MA, 2005.

[15] S. Stergiopoulos, "*Advanced Signal Processing Handbook : Theory and Implementation for Radar, Sonar and Medical Imaging Real-Time Systems*," CRC Press, Boca Raton, FL 33431, 2001.

[16] L.C. Godara, "*Smart antennas*," CRC Press, Boca Raton, FL 33431, 2004.

[17] 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 Communication Society on Sensor and Ad Hoc Communications and Networks, SECON '06*, pp. 518-526, September 2006.

[18] P.J. Vincent, M. Tummala and J. McEachen, "Optimizing the Size of an Antenna Array," *Proceedings of the 40th Asilomar Conference on Signals, Systems and Computers,* pp. 2281-2284, October 2006.

[19] J. Litva, and T.K.Y. Lo, *Digital Beamforming in Wireless Communications*, Artech House, Norwood, MA, 1996.

[20] B.D. Steinberg, *Principles of Aperture and Array System Design, including Random and Adaptive Arrays*, John Wiley & Sons, New York, 1976.

[21] B.D. Steiberg, "The Peak Sidelobe of the Phased Array Having Randomly Located Elements," *IEEE Transactions on Antennas and Propagation*, Vol. AP-20, No.2, pp. 129-136, March 1972.

[22] H. Ochiai, et al., "Collaborative Beamforming for Distributed Wireless Ad Hoc Sensor Networks," *IEEE Transactions on Signal Processing*, Vol.53, No.11, pp. 4110-4124, November 2005.

[23] Reichenbach Frank, et al., "A Distributed Linear Least Squares Method for Precise Localization with Low Complexity in Wireless Sensor Networks", *Proceedings of 2nd IEEE International Conference, DCOSS*, San Francisco, CA, pp. 514 -528, June 2006.

[24] C. Gkionis, "*Topology and Positioning of Wireless Sensor Network,*" Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2007.

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

[26] J.C. Chen, K. Yao, and R.E. Hudson, "Source Localization and Beamforming," *IEEE Signal Processing Magazine*, pp. 30-39, March 2002.

[27] V. Raghunathan, et al., "Energy-Aware Wireless Microsensor Networks," *IEEE Signal Processing Magazine*, pp. 40-50, March 2002.

[28] G. Golub and C.F. Van Loan, "*Matrix Computations*," The Johns Hopkins University Press, Baltimore, MA, 1996.

[29] S.J. Leon, "*Linear Algebra with Applications*," Prentice Hall, Upper Saddle River, NJ, 2006.

[30] T. Steihaug, and Y. Yalcinkaya, "*Asynchronous Methods and Least Squares: An Example of Deteriorating Convergence*," Technical Report No. 131. Department of Informatics, University of Bergen, Bergen, Norway, 1997.

[31] R.A. Renault, "A Parallel Multisplitting Solution of the Least Squares Problem," *Numerical Linear Algebra with Applications*, No.5, pp. 11-31, 1998.

[32] A. Berman, and R.J. Plemmons, "Cones and Iterative Methods for Best Least Squares Solutions of Linear Systems," *SIAM Journal on Numerical Analysis*, Vol.11, No.1, pp. 145-154, March 1974.

[33] J.J. Climent and C. erea, "Iterative Methods for Least-Square Problems Based on Proper Splittings," *Journal of Computational and Applied Mathematics*, No.158, pp. 43-48, 2003.

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

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Jeffrey B. Knorr
   Chairman, Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

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

5. Professor Roberto Cristi
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

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

7. Professor Weilian Su
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

8. Embassy of Greece, Air Attaché
   Washington,DC

9. Hellenic Air Force Academy
   Tatoi, Greece

10. Mike Niermann
    SPAWAR
    Charleston, South Carolina

11. Martin Kruger
    ONR
    Arlington, Virginia

12. Bernie Schneider
    SOCOM
    McDill AFB, Florida

13. Jamie Carson
    SRC
    Charleston, South Carolina

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

15. Richard Wylly
    SRC
    Charleston, South Carolina