A Turbo Approach to Distributed Acoustic Detection and Estimation

Sean Robert Egger

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Mechanical Engineering

November 17, 2009

Blacksburg, Virginia

Keywords: multi-sensor array, lossless information fusion, acoustic simulation

A Turbo Approach to Distributed Acoustic Detection and Estimation

Sean Robert Egger

ABSTRACT

Networked, multi-sensor array systems have proven to be advantageous in the sensor world. A large amount of research has been conducted with these systems, with a main interest in data fusion. Intelligently processing the large amounts of data collected by these systems is required in order to fully utilize the benefits of a multi-sensor array system. A robust but flexible simulation environment would provide a platform for accurately comparing current and future data fusion theories.

This thesis proposes a simulator model for testing fusion theories for these acoustic multi-sensor networks. An iterative, lossless data fusion algorithm was presented as the model for simulation development. The arrangement and orientation of objects in the simulation environment, as well as most other system parameters are defined by the user before the simulation runs. The sensor data, including noise, is generated at the appropriate time delay and propagation loss before being processed by a delay and sum beamformer and a matched filter. The resulting range-Doppler maps are modified to probability density functions, and translated to a single point of reference. The data is then combined into a single world model.

An iterative process is used to filter out false targets and amplify true target detections. Data is fused from each multi-sensor array and from each simulation run. Target amplitudes are gained if they are present in all combined world models, and are otherwise reduced. This thesis presents the results of the fusion algorithm used, including multiple iterations, to prove the algorithms effectiveness.

Acknowledgements

I would like to thank my graduate advisor, and committee chair, Dr. Michael Roan, for his support through graduate school. I appreciate the valuable advice he has given me, and the understanding that he has always shown me. He provided me with an opportunity to learn and grow, and the motivation to pursue it. I would also like to thank the rest of the Virginia Tech faculty, including Dr. Dennis Hong and Dr. Martin Johnson on my committee, for guiding me through a memorable and fulfilling college career.

Thank you to Elizabeth Hoppe and Maximilian Opheim for being true friends. Liz was always willing to answer my questions and offer help, regardless of how often I needed it, and made lab a fun place to work each and every day. Max has been, and always will be a true friend, and it's an honor to know him. Thanks for everything my friend, slowly but surely the merger continues!

My parents, Robert and Deborah Egger, deserve more appreciation than I have words for. Their guidance has prepared me for the world, while their never-ending encouragement allowed me to always be myself. I lead a blessed life, and I owe it all to my parents. I would also like to thank the rest of my family for their love and support, including Cory, the best brother anyone could hope for.

Most importantly, I would like to thank my wife Bree-anne. I would not be where I am today without her next to me. Her support as a friend, and her love as a wife have pushed me to be the best man I can be. She is the best woman I have ever met, and I look forward to the rest of our lives together.

And of course, thank you to Branagan, my son. You are my whole world. Thank you for the daily playtime sanity breaks from a busy life. You amaze me.

Table of Contents

List of Figures

List of Tables

List of Variables

∇	Laplace operator
p	pressure
c	speed of sound
t	time
d	density
k	adiabatic index
T	temperature
r	*propagation* distance
s	number of samples
f_s	sampling frequency
v_{trans}	speed of transmitter
v_{rec}	speed of receiver
f	frequency of the signal
Δf	perceived frequency shift
$s(t)$	original signal
w_i	weight value
τ_i	time delay
m	lag
$x(t)$	received signal and noise

T	signal duration
A	peak amplitude
x_0	curve center in x-axis
y_0	curve center in y-axis
σ_x	variance of curve in x-axis
σ_y	variance of curve in y-axis

Chapter 1:

Introduction

This chapter will serve as an introduction A Turbo Approach to Distributed Detection and Estimation. This introduction will include a summary outline of the thesis, followed by an explanation of the motivation for this thesis and the relevance to research.

1.1 Thesis Outline

This chapter discusses the motivation and significance of the research presented in this thesis. Chapter 2 provides a brief background on the critical components of multi-sensor array fusion for sonar. These components include sonar fundamentals, sound propagation, special processing, matched filtering techniques, and data fusion theory. Also provided in Chapter 2 are current and future applications for this technology. In chapter 3, a detailed discussion of the simulation code is presented beginning with an overview of the simulation code structure. The parameters and assumptions used in the simulation code are also established in Chapter 3. Each subsequent section of Chapter 3 presents detailed descriptions of the simulator modules. The purpose of each module is described, and graphical results are presented to validate the modules effectiveness and operation. The effects that the user defined parameters have on the output of the modules is also presented in these sections. A summary and conclusions of the work are discussed in chapter 4 of this thesis. Chapter 5 will wrap up the thesis by presenting suggestions for future advancement of the research.

1.2 Motivation and Relevance of Research

There has been a large volume of work performed in multi-sensor, networked arrays in the past several years [1-22]. A networked system of multi-sensor arrays has many advantages. Multi-sensor arrays are innately robust against imprecision of individual sensors because post processing combines the data from all the sensors in the array [13]. Because of this, the cost associated with multi-sensor systems can be lower than that of single sensor systems where high fidelity hardware is required. The lower cost of the sensors also translates to more readily available hardware, meaning quick deployment in time sensitive applications. Sensor arrays can be used to accurately determine the direction and velocity of a target of interest due to the separation of sensor nodes. They can also be effectively used to gather information about very large areas of interest.

Currently, the main interest is in fusion of the sensor-level information collected by each sensor array [17]. This is an advancement over fusion of post-processed data (such as track fusion), which has an associated loss of information inherent in the formation of tracks [11]. There is desire for lossless information fusion methods that are robust to changes in environmental and situational factors, while also staying computationally feasible in real world situations. A primary motivation for this thesis was the paper "Lossless Information Fusion for Active Ranging and Detection Systems", by Sibul, Roan, Schwartz, and Coviello, which discusses a data fusion technique for multi-sensor array networks [17]. This paper served as a basis for the simulator algorithm model used in this thesis.

To accurately compare the many data fusion techniques currently in development, a fully controlled but flexible testing environment is needed. In order to achieve this environment, a robust simulator was developed. This thesis explores the development of that simulator code, including example simulation results and a discussion of parameter effects.

Chapter 2:

Background

Multi-sensor, networked arrays can be used to measure any wavefield type depending on the particular type of sensors used. This thesis, however, will focus on acoustic wavefields and the post processing of acoustic signals using distributed sonar signal processing methods. The software discussed in this thesis simulates 2-dimensional acoustic testing environments. Adaptation of the software to simulate other types of wavefields (such as electromagnetic for distributed radar applications) or calculate 3-dimensional environmental models is possible, and will be discussed in the final chapter of this thesis. Due to the focus on acoustics, this section will present necessary background information on acoustic processing. This discussion will cover sound propagation characteristics and sonar, including different types of sonar and the methods for measuring signals. An explanation of beamforming and matched filtering techniques will also be presented, with a focus on the versions used in this simulation code.

2.1 Sound Propagation

Acoustic waves are longitudinal pressure waves that propagate radially outward from the sound source [14]. Sound waves can be modeled using the linear equation for longitudinal waves presented below

$$\nabla^2 p = \frac{1}{c^2}\left(\frac{d^2 p}{dt^2}\right) \qquad \text{(eq. 2.1)}$$

where ∇ is the Laplace operator, *p* is the pressure, *c* is the speed of sound, and *t* is time. This equation represents acoustic wave propagation in two dimensions, which is sufficient for all simulation calculations discussed in this thesis.

Sound propagation is dependent on the signal frequency, the propagation medium, and environmental factors. Sound requires matter to propagate, and the speed of sound is dependent on the medium's modulus of elasticity and density [14]. The example simulations in this thesis assume air as the propagation medium. The speed of sound in air is dependent on the temperature of the air, and can be calculated using the equation

$$c = \sqrt{\frac{p}{d}k} \qquad \text{(eq. 2.2)}$$

where p = pressure of air, d = density of air, and k equals the adiabatic index or ratio of specific heats. Since the ratio of pressure to density of air is a constant, the equation can be approximated to

$$c = 331.5 + 0.60T \qquad \text{(eq. 2.3)}$$

where T is the temperature in Celsius. The speed of the sound waves is determined by the user prior to the simulation.

When considering multi-sensor acoustic arrays, the sound field can be classified into three different range-dependent regions. These regions are known as near field and far field, with the area in between called the transition area. Sound propagation characteristics are difficult to simulate and analyze in the near field because small changes in distance and orientation of objects can result in large changes in sound pressure levels. The distance to the transition area is dependent on the specific characteristics of the microphone array used [7]. For all simulations in this thesis, the width of the microphone array is considered significantly smaller than the distance of transmission, and it is assumed all received signals are in the far field. This is typically referred to as plane wave propagation.

Because all sources in the simulation are assumed to be far field sources, further assumptions can be made about the wavefields received by the microphone arrays from these sources. When the transmission distance is significantly larger than the sensor spacing in the multi-sensor array, then across the width of the array, the curve of the wavefield due to spherical propagation is considered negligible. The sound waves are assumed planar and parallel to each other for all calculations in this thesis. This assumption is referred to as the plane wave approximation and is common in acoustic analysis. This assumption also dictates that if the direction of the multi-sensor array is exactly normal to the incoming wavefield, then all sensors in the array will receive the signal simultaneously. A figure demonstrating this assumption can be seen below.

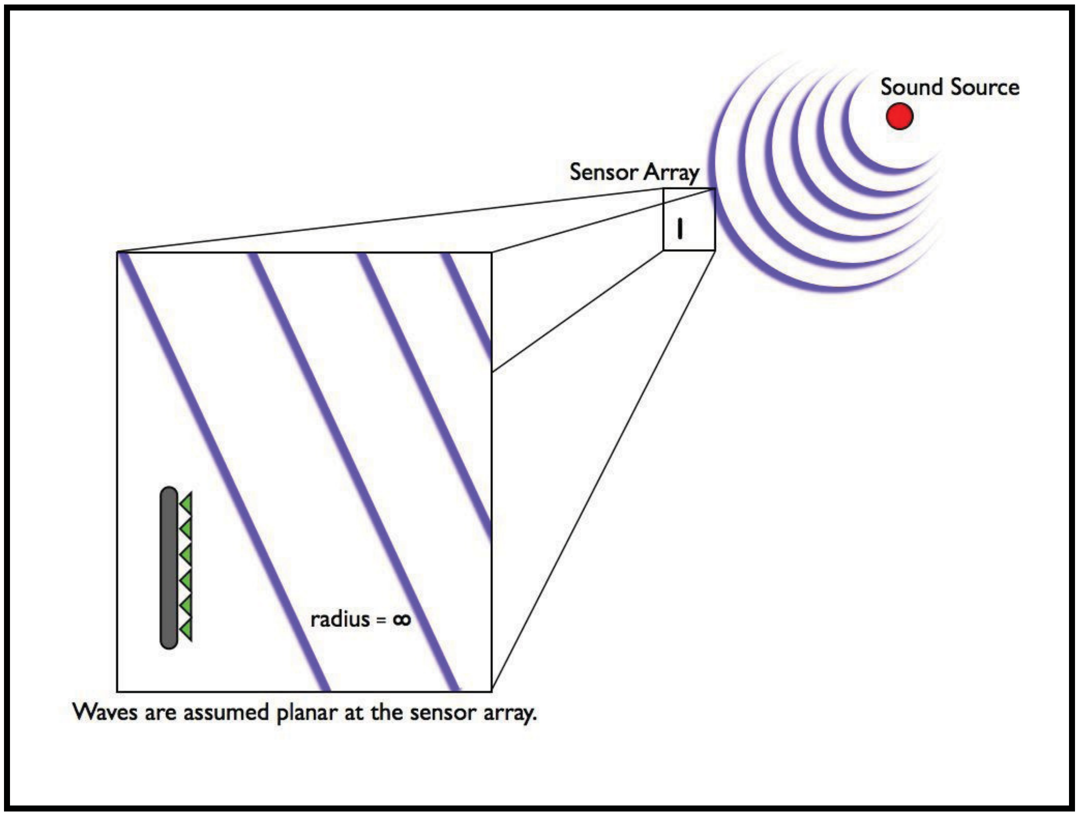

Figure 2.1. Demonstration of the planar wave assumption. If it is assumed that the array is in the farfield and that the width of the array is significantly less than the distance of transmission, then the sonar waves can be assumed planar at the sensor array.

The energy of sound decays with distance and is dependent on the frequency of the signal, the medium, and the propagation environment. The energy of the sound waves is absorbed by the medium and into the surrounding surface areas as the sound reverberates. This is especially true of higher frequency and ultrasonic sound waves [14]. Assuming a point source transmitter in a 3-dimensional environment, the sound waves transmit spherically outward and the signal intensity can be generalized by the Inverse Square Law [14]. The Inverse Square Law of sound dictates that the intensity of the sound is inversely proportional to the distance the sound has travelled, or

$$I \propto \frac{1}{r^2} \qquad \text{(eq. 2.4)}$$

where I is the intensity of the sound wave per given area, and r is the distance travelled. Complex algorithms are used to represent additional sound propagation loss factors in real world experiments. However, the simulator code uses a simplified sound propagation-loss model.

2.2 SONAR

Sonar, which stands for SOund Navigation And Ranging, is a sensing method that uses acoustic waves to gather information about a target or environment. There are two forms of sonar sensing, active and passive [9]. Passive sonar uses a receiver to collect information about signals of interest. Target detection and classification can be achieved by identifying particular acoustic signatures in the received signal. One main challenge with passive sonar systems is that the characteristics of the incoming signal are typically unknown. This has necessitated the development of complex classification algorithms to distinguish signals of interest from background clutter [15].

Active sonar systems use a transmitter to emit a specific signal, or ping, into an environment and then listen for echoes of that signal to reflect back to the receiver. There are three types of active sonar: monostatic, bistatic, and multistatic. An active system is classified as monostatic when the active source and the receiver are co-located. A bistatic system has a single active source and a single receiver separated in space, though typically the active source is used as a pinger and a receiver. Multistatic systems have multiple active sources and/or receivers that are also separated in space [14]. The simulator developed for this thesis is customizable for any number and arrangement of active sources and receivers, however at least one active source must be present.

Active sonar ranging is achieved by measuring the time delay between when the signal is transmitted and when the echo reaches a given receiver. Typically, sonar is used in underwater environments because sound waves propagate easily underwater. However, the example simulations presented in this thesis assume in air conditions, so the calculations shown will be performed using the speed of sound waves traveling through air in ideal conditions at 343 meters per second. Assuming this, and the time delay between transmission and reception, the distance to the target can be calculated using

$$d = \frac{t}{c} \qquad \text{(eq. 2.5)}$$

where t is the time of transmission, and c is the speed of sound. If the number of samples until reception is known, as is the case in the simulated environment discussed in this thesis, the equation above can be modified as

$$d = c\frac{s}{f_s} \qquad \text{(eq. 2.6)}$$

where s is the number of samples before the ping is received, and f_s is the sampling frequency used in the simulation. Assuming a sampling frequency of 5000 Hz and that the ping was heard after 7500 samples, the distance to the target can be calculated as follows

$$d = c\frac{s}{f_s} = 343m/s\left(\frac{7500samples}{5000Hz}\right) = 514.5m$$

The bearing of the target can also be determined when multiple receivers are using together with each other in a multi-sensor array due to the time delay between received signals for each of the sensors. The process used in this thesis to analyze the bearing is called beamforming and will be discussed later in this chapter.

The simulator assumes stationary transmitters and microphone arrays. However an understanding of the Doppler Effect is required to understand the matched filtering and fusion algorithms used. If the target is not stationary, then the velocity can also be estimated by exploiting the Doppler effect. The Doppler effect is the change in frequency of a signal due to the motion of the source or receiver. As a source or receiver move with respect to the emitted wavefields, the waves undergo an expansion or compression, resulting in the receiver measuring a perceived lower or higher frequency respectively [9]. A figure demonstrating the Doppler effect can be seen below.

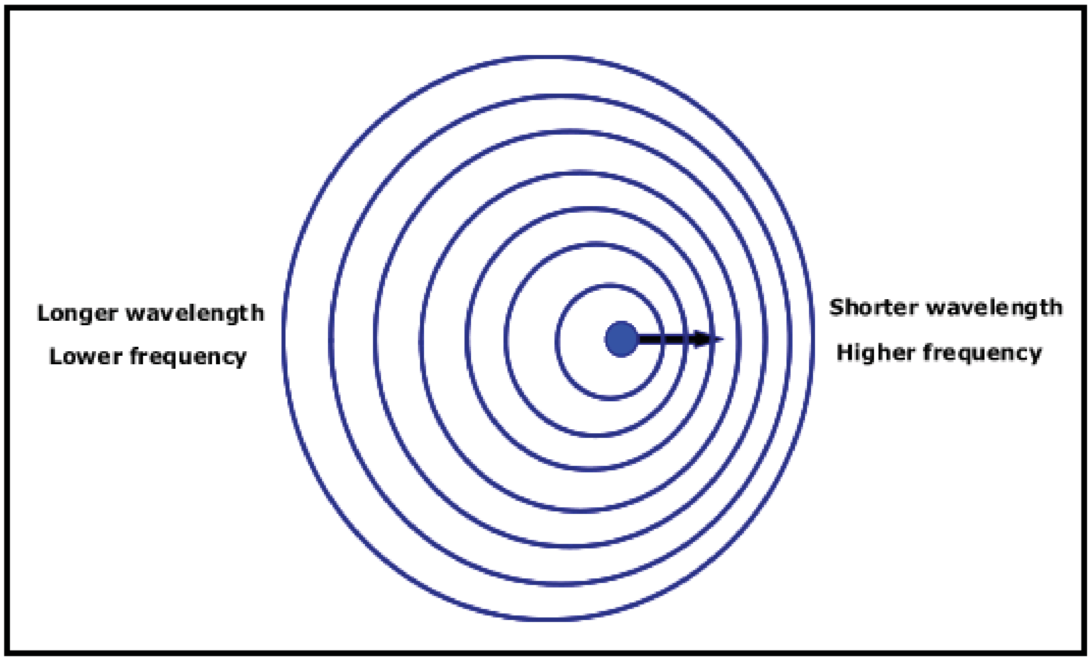

Figure 2.2. Demonstration of the Doppler affect. The motion of the sound sources compresses or expands the sound waves, resulting in a measured frequency shift at the receiver.

The change in frequency is referred to as a Doppler shift, and can be calculated using the equation

$$\Delta f = f\left(\frac{v_{trans} - v_{rec}}{c}\right) \qquad \text{(eq. 2.7)}$$

where v_{trans} and v_{rec} are the speeds of the transmitter and the receiver respectively, f is the frequency of the unaltered signal, c is the speed of sound through the medium, and Δf is the perceived frequency shift. Note that for situations where the motion of the transmitter and receiver is angular with respect to each other, the velocities used in the equation are the normal velocity components.

In an active sonar system where the transmitted signal is known, the velocity of the target can be determined. A comparison is made between the signal received by the sensor array and several frequency-shifted versions of the known signal, and the relative velocity of the target can be calculated.

2.3 Beamforming

As discussed earlier in this chapter, it is assumed that as sound waves propagate radially outward from the signal source, they reach the receiver in the far field as a plane wave. If the orientation of multi-sensor array is not normal to the direction of the sound source, then the distance between each individual microphone and the sound source will vary along the array. Figure 2.3 below illustrates the geometry.

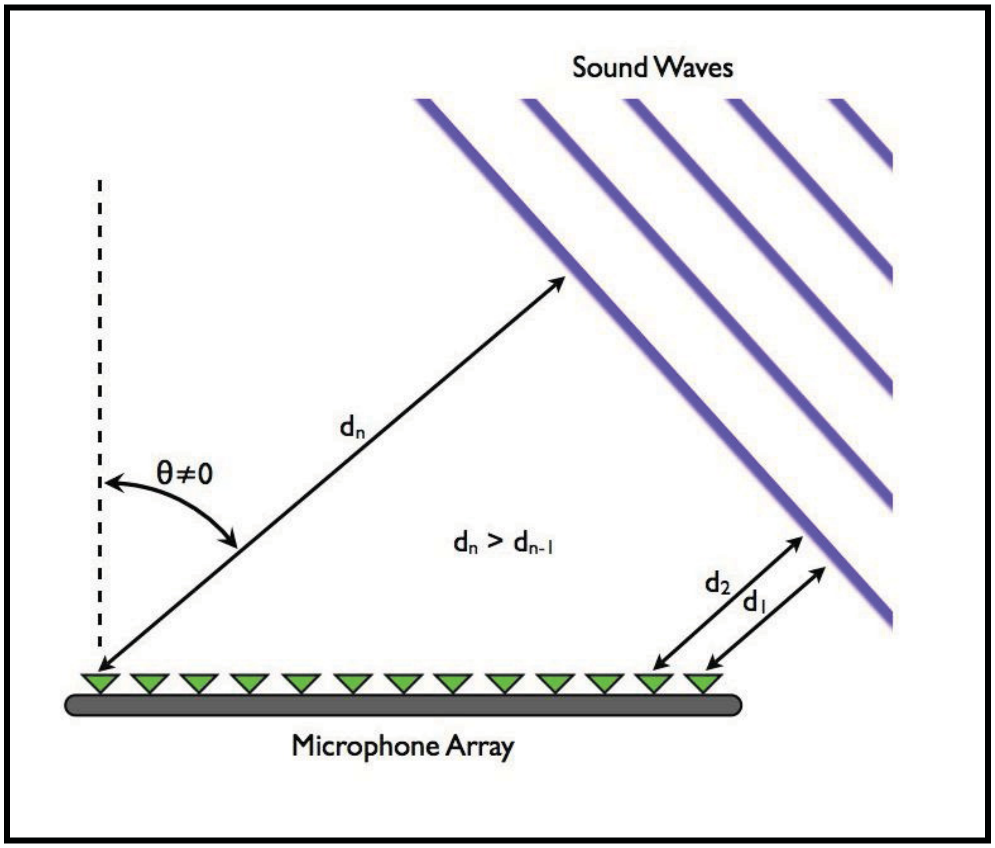

Figure 2.3. Demonstration of time delay between sensors due to orientation. If the microphone array is not oriented parallel to the planar sound waves, then the individual sensors receive the signal at different time delays.

This difference in distance translates to a difference in the time of arrival of the signal at each microphone in the array. Using this phase delay, a bearing estimate of the signal source can be made using a process called beamforming. A beamforming algorithm uses a calculated set of frequency dependent weights applied to each channel of the array to focus the array towards a specific angle. The resulting weighted channels are then summed together to provide a single output per angle of interest, or steering angle. Selecting the correct set of weights will phase align all of the channels if the beamformer is steered toward the target and therefore coherently sum the signal at the angle of interest for each channel. Signal sources at other angles result in out of phase summation and are reduced.

There are several beamforming algorithms in use today, categorized as fixed or adaptive. Fixed beamformers are signal independent, while adaptive beamformers use properties of the incoming signal during processing to improve performance. Another classification of a beamformer is whether they are narrowband or broadband. Narrowband beamformers are valid for only a single frequency of interest while steering the array. Beamforming at a specific frequency other than that of the incoming signal frequency can result in significant signal distortion. Broadband beamformers involve more complex algorithms to achieve beamforming across the desired spectrum of frequencies without distorting the signal of interest [12].

The simulation developed for this thesis uses a conventional, narrowband, delay and sum beamformer. This beamformer uses a Fourier transform to translate the raw data into the frequency domain before applying the appropriate weight vector. Once the vector is applied, the data is summed and transformed back into the time domain. While the beamformer processes the data collected by each microphone in the multi-sensor array, the output is a single arriving from the angle of interest. The beamformer output is described by the equation

$$B(t) = \sum_{i=o}^{N-1} w_i s_i (t - \tau_i) \qquad \text{(eq. 2.8)}$$

where $s(t)$ is the signal received by the microphone, w_i is the weight applied to the microphone, and τ_i is the time delay.

2.4 Matched Filtering

A matched filtering process compares the known transmitted signal with the received, beamformed signal to detect the presence of the known signal. The matched filter cross-correlates the beamformer output for a given steering angle with hypothesized versions of the transmitted signal that are both time shifted as well as Doppler shifted [21]. The cross-correlation function for the matched filter is

$$R(m) = \frac{1}{T} \int_0^T s(t) x(t+m) dt \qquad \text{(eq. 2.9)}$$

where m is the cross-correlation lag, T is the signal duration, $s(t)$ is the original signal, and $x(t)$ is the combined signal and noise. The matched filter produces a 2-dimensional matrix of values for the microphone array at the given steering angle of the beamformer. This matrix represents the amplitude of the received signal per given range and Doppler shift, and is referred to as a range-Doppler map. These range-Doppler maps are treated as sufficient statistics for each array at the angle of interest and form the basis for the world models developed by the simulator.

2.5 Current Applications of Microphone Arrays

Multi-sensor, networked arrays such as microphone arrays are currently being used in a variety of ways, and new applications are continuously being developed for the near and distant future. Military defense organizations and Homeland Security are some of the primary users of multi-sensor networked arrays today [16]. Networked arrays have proven very advantageous for underwater tracking such as torpedo defense systems on submarines and mine searching [1]. These array systems provide a large area of sensing coverage while also effectively providing robustness against false detections. Multi-sensor arrays have also proven useful in applications where multiple sensors are required for triangulation such as in sniper locating systems. Anti-terrorism technologies are also currently in high demand and research is currently being conducted on how to further apply networked multi-sensor arrays to the field [16].

Networked, multi-sensor arrays can be easily deployed in a large variety of environments, and therefore many additional potential applications for these systems exist. When human interaction needs to be minimized due to inhospitable conditions, such as a nuclear waste site, these systems can be set up quickly and require little to no interaction during data collection. Disposable data collection systems for highly dangerous situations are also feasible with multi-sensor arrays due to their low cost. Multi-sensor arrays are also ideal candidates for very large environments, such as deep sea and outer space, because large coverage areas can be achieved by simply adding additional multi-sensor arrays to the system. Search and rescue missions in particular are dependent on quick deployment and a large coverage area, and can profit from the user of multi-sensor array systems.

Chapter 3:

Simulation Development

3.1 Overview of Simulation Code

This section provides a detailed overview of the simulator software architecture and development. This overview will assist in understanding the specific objectives of each module in the simulator code. Specific details of the simulator modules will be presented later in this chapter. The figure below shows a flowchart of the simulator software architecture.

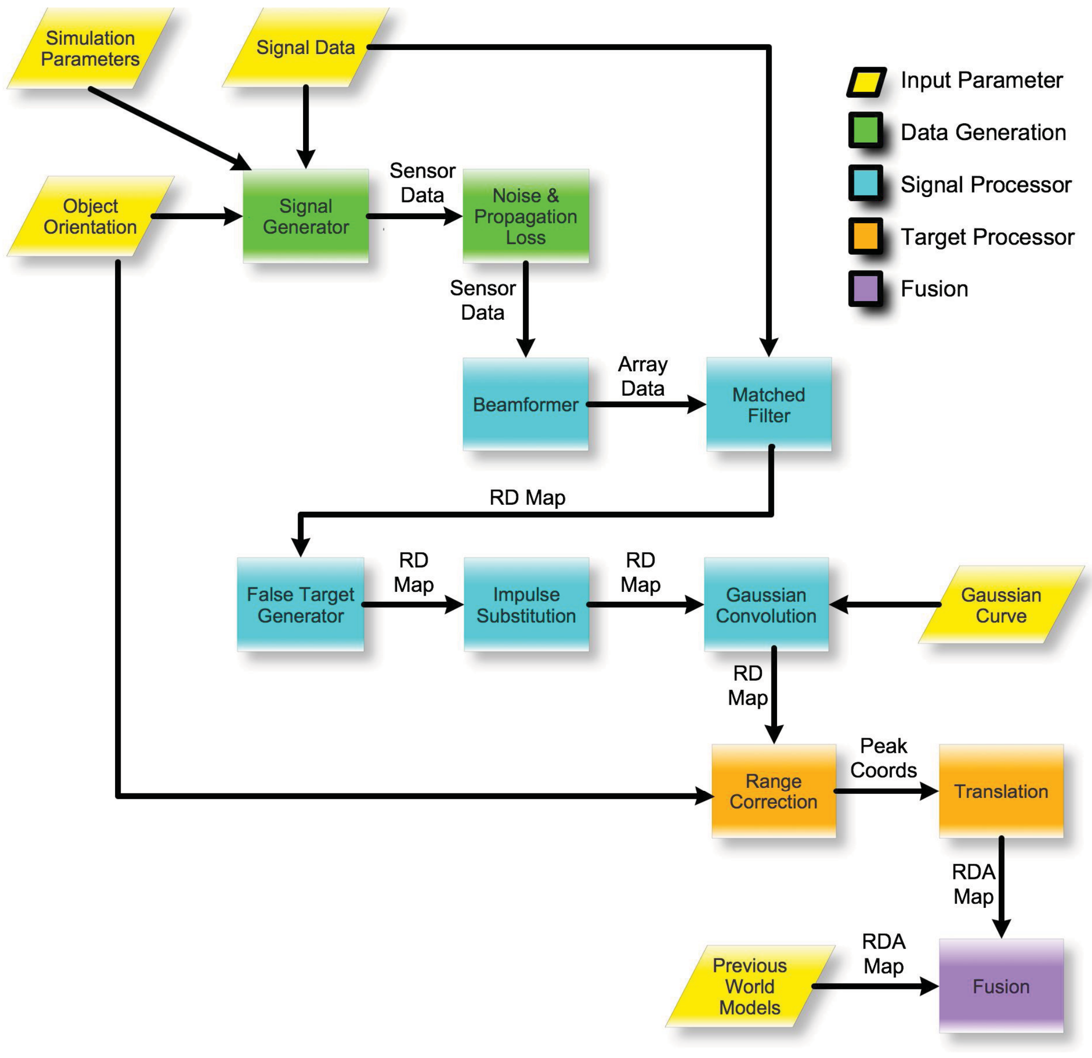

Figure 3.1. Simulator Code Flow Chart. The simulator modules are divided up into 4 categories based on the type of information they process. This flow chart shows those categories with color.

The software simulator can be divided into four main sections, based on the type and level of data that is processed by that section. These sections are illustrated in Figure 3.1 above by color. The first section is referred to as Data Generation, and it develops the data for each multi-sensor array in the simulation. The Signal Processor section, is called once for each angle step of each multi-sensor array, and is therefore lower level than the Data Generation section. A third section, the Target Processor, is the lowest level section because it handles each detected target for each angle handled by the Signal Processor, and is called more times per simulation than any other section. The final section, the Fusion section, has the highest level as it is only called once for each simulation run and processes all multi-sensor data at once. The table below demonstrates the typical number of times a section will be called in single iteration of the simulator.

Section	**Type of Data Processed**	**Typical Step Size**	**Number of iterations called**
Data Generation	array	3	3
Signal Processing	angle	180	540
Target Processing	target	35	18900
Fusion	world model	1	1

Table 3.1. Chart shows the number of times that each module category gets called in a single simulation iteration. The number of times they are called is dependent on the type of data to be processed.

The Data Generation section can be further broken down into two modules. The first module, referred to as the *Signal Generation* module builds the signal received by each sensor in each array at the appropriate phase for the sensor geometry and orientation. The second module increases the realism of the simulated signal from the previous module by simulating noise and propagation loss. This module is referred to as *Noise and Propagation Loss Simulation*. After being processed by this module, the phase delayed signal matrices are also zero-padded to simulate echo distance from the signal source to the receiving sensor via the target reflection. The output of the Data Generation section is the simulated raw data that would be collected by each sensor array in a real world experiment.

There are five modules that make up the Signal Processing section. Each of these modules is called once per angle of interest for each multi-sensor array. The first module, called the *Beamforme*r, uses narrowband, delay and sum beamforming to increase the signal to noise ratio of signals arriving from a specific angle of interest. The beamformed data is then sent to the *Matched Filter* module where a range-Doppler map is generated for each angle step. To increase realism and further demonstrate target localization ambiguity, false targets are added to the range-Doppler map in the *False Target Generator* module. This new range-Doppler map is then passed to the next module called *Impulse Substitution*. It is thresholded and each remaining peak is replaced by a single impulse of equal amplitude. The final module of the Signal Processing section is called *Gaussian Convolution*. This module performs a matrix convolution with a 2-dimensional Gaussian curve, resulting in a probability density map of target range versus Doppler value.

The next simulator section processes each detected target in the probability density map individually. The detected range of the target in the probability density map needs to be recalculated to compensate for echo distances. The module that calculates this is called *Range Correction*. Once the true target range has been determined, the target location with respect to the environment origin is calculated for each target in the map using the *Translation* module.

The origin-translated maps are then stacked into one 3-dimensional matrix for each sensor array. Each of these matrices consists of target range versus Doppler versus angle, with each element value representing signal amplitude. These 3-dimensional maps are called "array models" as they represent all processed data received by each multi-sensor array.

The Fusion section has one module, also called *Fusion*, which combines the array models for each sensor array into one model, referred to as the "world model". This a posteriori world model then becomes the a priori model for the next simulation iteration.

3.2 Establishment of Variables and Assumptions

Initialization of the simulation begins with a set of simulation parameters being established by the user. This section discusses the variables required by the simulator, the level of control the user has on the variables, and what effect changing some of those variables has on the simulation results. This section will also establish any major assumptions in the simulator development.

The simulator environment is a 2-dimensional area of any size determined by the user, with an origin defined at the bottom left corner when looking from an overhead view. The positive x-axis is defined as 0° from the origin, and rotation is positive in the counter-clockwise direction. For all demonstrations in this thesis, a 200m by 200m environment is used. Figure 3.2 below establishes this simulator coordinate system.

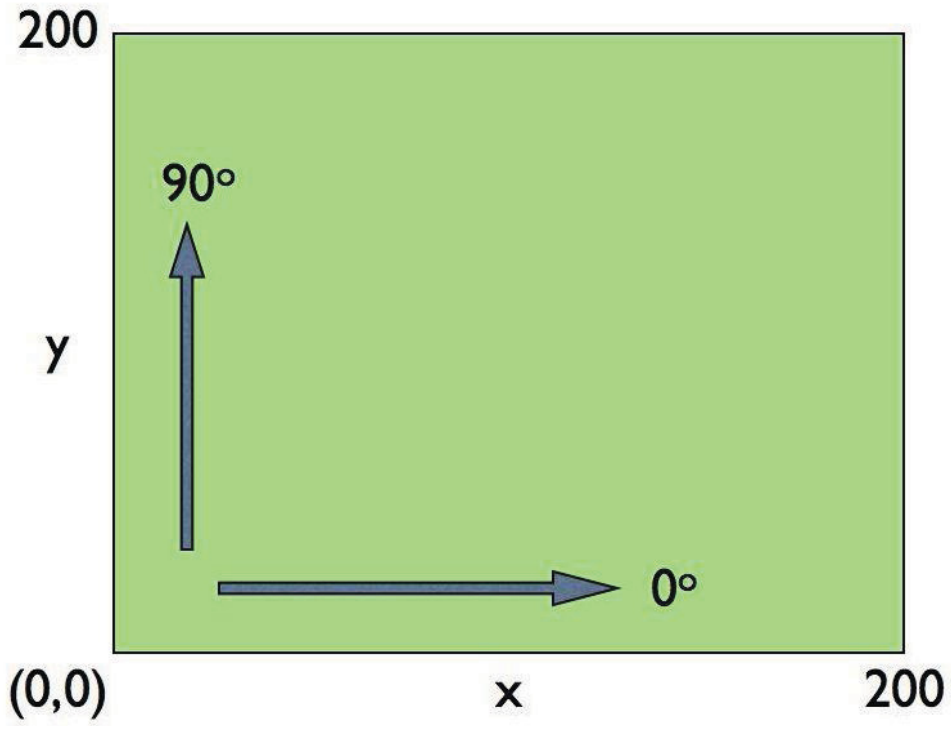

Figure 3.2. Simulation environment coordinate system. The simulation environment is 200m by 200m with the origin in the bottom, left corner. The x-axis is 0 degrees with positive rotation being counter-clockwise.

The first set of parameters established by the user set up the initial location and orientation of objects in the simulated environment. In real world applications, the locations and orientations of these objects are very rarely completely controllable. Often, equipment arrangement is subject to the type of equipment used, the desired target type, and environmental limitations. For this reason, flexible orientation geometry in a simulator is desirable. The number of sub-arrays to be used can be defined by the user, and for each array there are 4 user-defined variables. These include the number of sensors in each array, the x and y-coordinates of the array, and the orientation angle of the array. The array element spacing is also user-defined, and is the same for all sub-arrays in the simulation. The flexibility of these parameters allows the user to replicate existing sensor equipment in the simulation. Also, controlled comparisons can be performed between differing experimental set ups using these parameter controls.

For example simulation results generated in this thesis, three microphone arrays are used, with 8 microphones per array, a spacing of 0.75 inches, arranged and oriented according to the following table.

Array	Number of Elements	Element Spacing (in)	X Coord. (m)	Y Coord. (m)	Orientation (degrees from x-axis)
1	8	0.75	10	10	45
2	8	0.75	50	150	270
3	8	0.75	180	80	180

Table 3.2. Table shows some of the user-defined parameters used in most of the example simulation results in this thesis. The parameters for each microphone array are listed.

The x- and y-coordinates of signal sources and targets in the simulation have to be established as well. Orientation angle for sources and targets is not required because they are always assumed to be omni-directional in the simulator. For example simulation results used in this thesis, one signal source and one reflector target are used, arranged according to the coordinates in the following table.

Object	X Coord. (m)	Y Coord. (m)
Target	100	100
Source	80	20

Table 3.3. This chart shows some of the user-defined parameters used for most example simulation results in this thesis. The parameters for the target and sound source are listed.

A graphical representation of this example simulation environment can be seen in Figure 3.3 below.

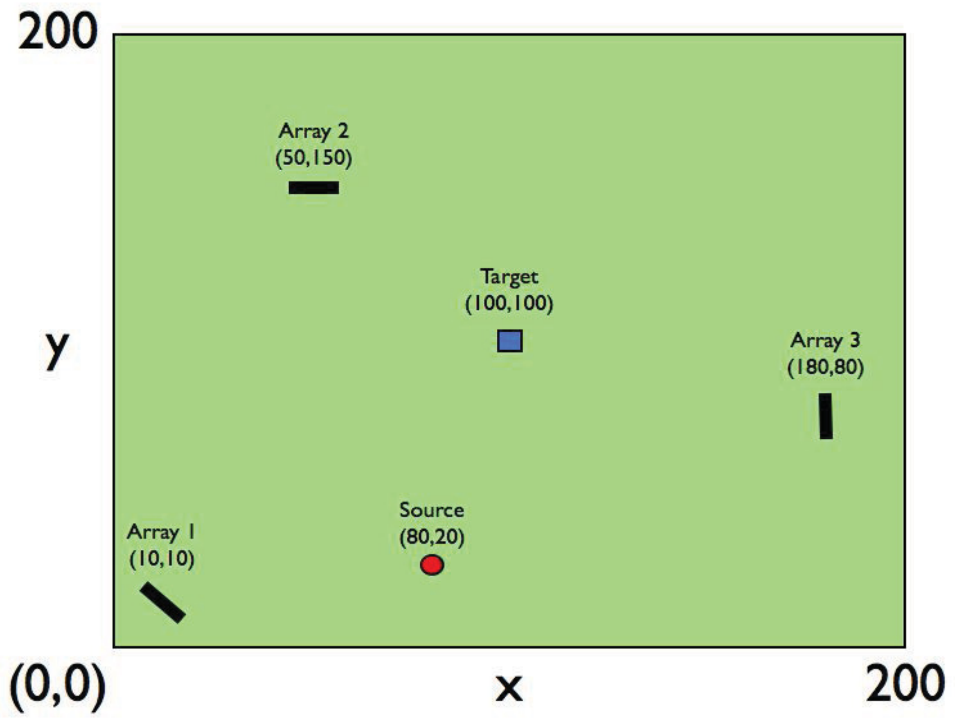

Figure 3.3. This figure shows the arrangement and orientation of the objects used in the example simulation for most results in this thesis. The numerical information can be found in Table 3.2 and Table 3.3.

Several other parameters must be established by the user in order to define how data processing should be executed. Both the signal characteristics as well as the sampling frequency can be individually set, though careful consideration should be taken to ensure the sampling frequency is greater than or equal to the Nyquist rate of the chosen signal, or

$$f_s > f_n = 2\beta \qquad \text{(eq. 3.1)}$$

where β is the highest frequency found in the signal with a non-zero energy. The Signal Generator can take any signal type as an input, though generally a single tone, a chirp, or white noise are used. The signal reaches amplitude and ends instantaneously and does not ramp to or from the desired frequency. The signal must be fully defined by the user, including the signal amplitude, frequency range, and length.

Figure 3.4 below shows spectograms of three different signals after being processed by the Beamformer module. The chirp signal increases from 1000 Hz to 2000 Hz, the tone is at 1500 Hz, and the white noise is Gaussian. All three signals last for 0.25 seconds, and there is no noise in the simulation.

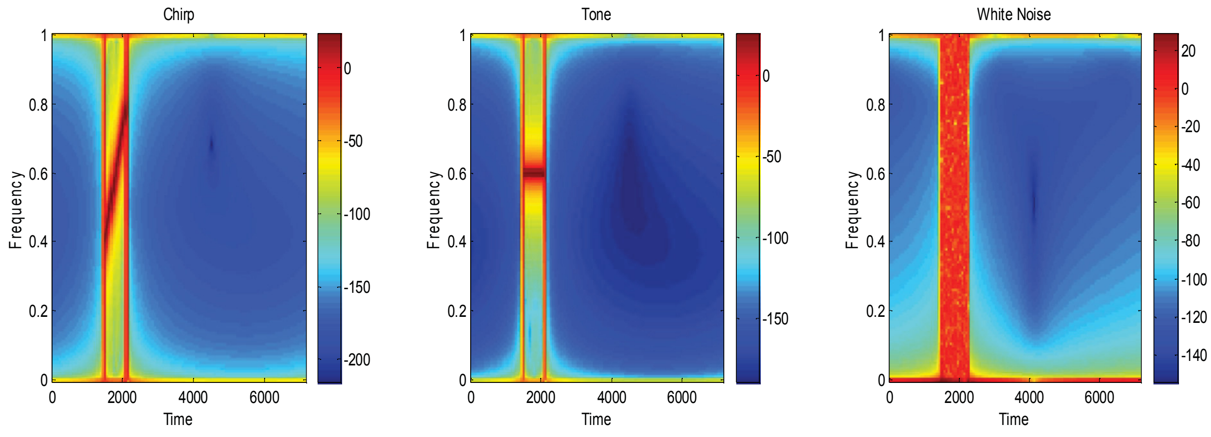

Figure 3.4. Spectograms of the Beamformer module output using various signal types. The graphs show a chirp signal, a tone, and white noise from left to right.

For all three signal types, the fusion algorithm is successful in determining the true target location. The signal type does not effect the number if iterations required to make a determination.

Noise is generated in the simulator according to a desired signal to noise ratio that is determined prior to the simulator running. Using the user-defined signal amplitude and desired signal to noise ratio, the average noise amplitude is calculated and applied to the signal with the following equation.

$$\left(\frac{A_{signal}}{A_{noise}}\right)^2 = SNR \qquad \text{(eq. 3.2)}$$

where A is the RMS amplitude and SNR is the signal to noise ratio. Solving for noise amplitude gives the equation

$$A_{noise} = \frac{A_{signal}}{\sqrt{SNR}} \qquad \text{(eq. 3.3)}$$

The user can opt for an infinite signal to noise ratio indicating no noise, or to have any level of noise present in the system. Lower signal to noise ratios result in more difficult detection of the target echo. This is desired to fully demonstrate the abilities of the fusion method being tested in the simulator. The table below shows typical sampling, signal, and noise parameters for a simulation run. These parameters will be used and referenced for many examples in this thesis.

Signal Type	Signal Length (sec)	Signal Freq (Hz)	SNR	Sampling Freq (Hz)
Chirp	0.25	1000-2000	0.5	5000

Table 3.4. Table shows some of the user-defined parameters used in most of the example simulation results in this thesis. The parameters for the signal, sampling, and noise are shown.

The following figure illustrates the effect that the chosen signal to noise ratio has on the generated signal. The first graph shows a chirp signal with no noise effects. The other graphs in Figure 3.5 show increasingly higher signal to noise ratio values, demonstrating the effect noise has on obscuring the signal.

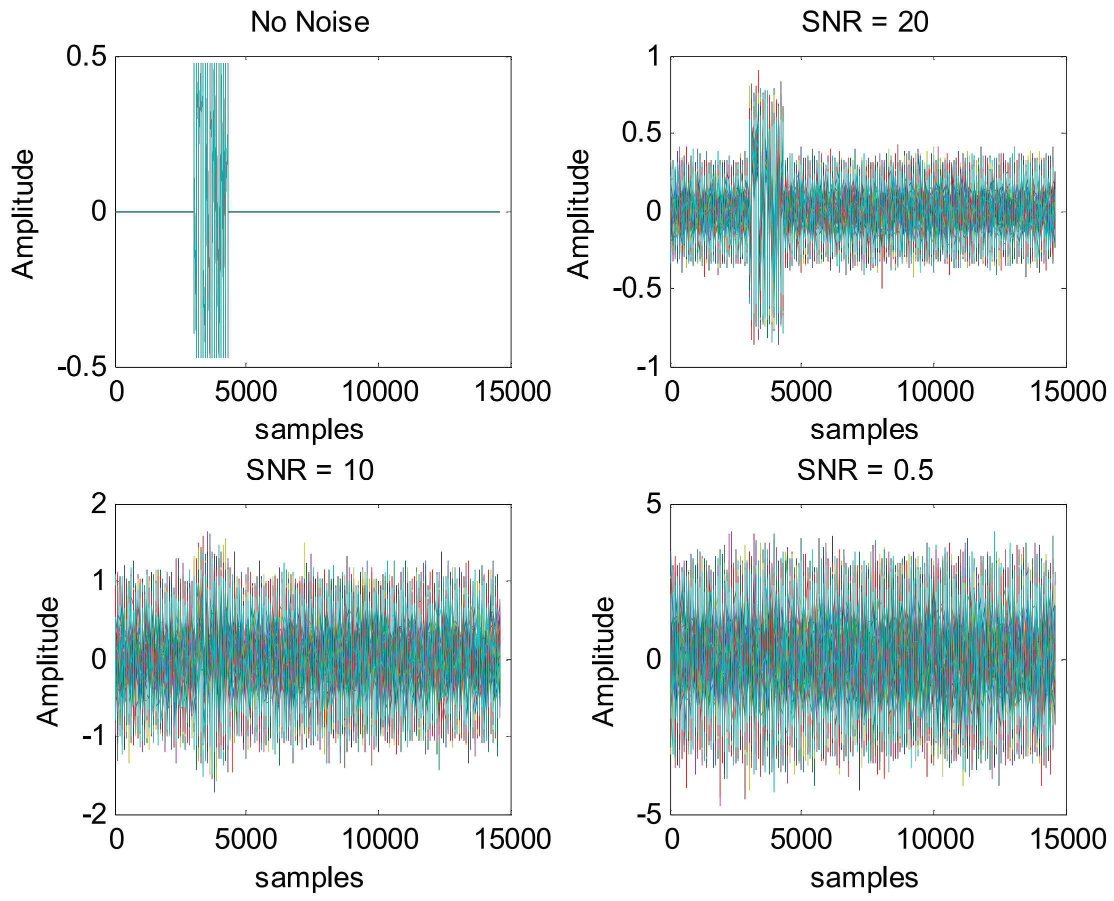

Figure 3.5. Output of the Data Generation section with various SNR values. From left to right and top to bottom, the figures show SNR values of infinity, 20, 10, and 0.5.

As the signal to noise ratio is decreases and the ambiguity of the signal in the data is increased, more iterations of the fusion algorithm are required in order to determine the correct target location.

Calibration constants are also defined by the user before simulation. The calibration constants are included in the parameters to add flexibility to the simulation and to allow the code to be adapted for real world experimentation. When using real world sensors, they typically have differing physical characteristics that can significantly affect the sensor output in terms of gain and relative phase. The calibration constants correct for these differences and ensure more accurate measurements. For a fully simulated environment, all zeros are generally used as calibration constants. However, these constants could be adjusted to simulate calibration errors and non-ideal data for specific research.

In the simulation, a convolution with a 2-dimensional Gaussian curve is performed in the generation of probability density maps. The Gaussian curve is established during the initial call of parameters and the size and variance of the curve can be easily controlled by the user. The 2-dimensional Gaussian curve is defined by the function

$$f(x,y) = Ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2}\right)} \qquad \text{(eq. 3.4)}$$

where A is the amplitude of the peak, x_0 and y_0 are the center of the Gaussian curve, and σ_x and σ_y are the variance in the x and y directions.

Figure 3.6 below presents graphics using two different Gaussian variance values. The first figure for each variance value shows the 2-dimensional Gaussian curve, while the second figure shows the range-Doppler map after the *Gaussian Convolution* module processing.

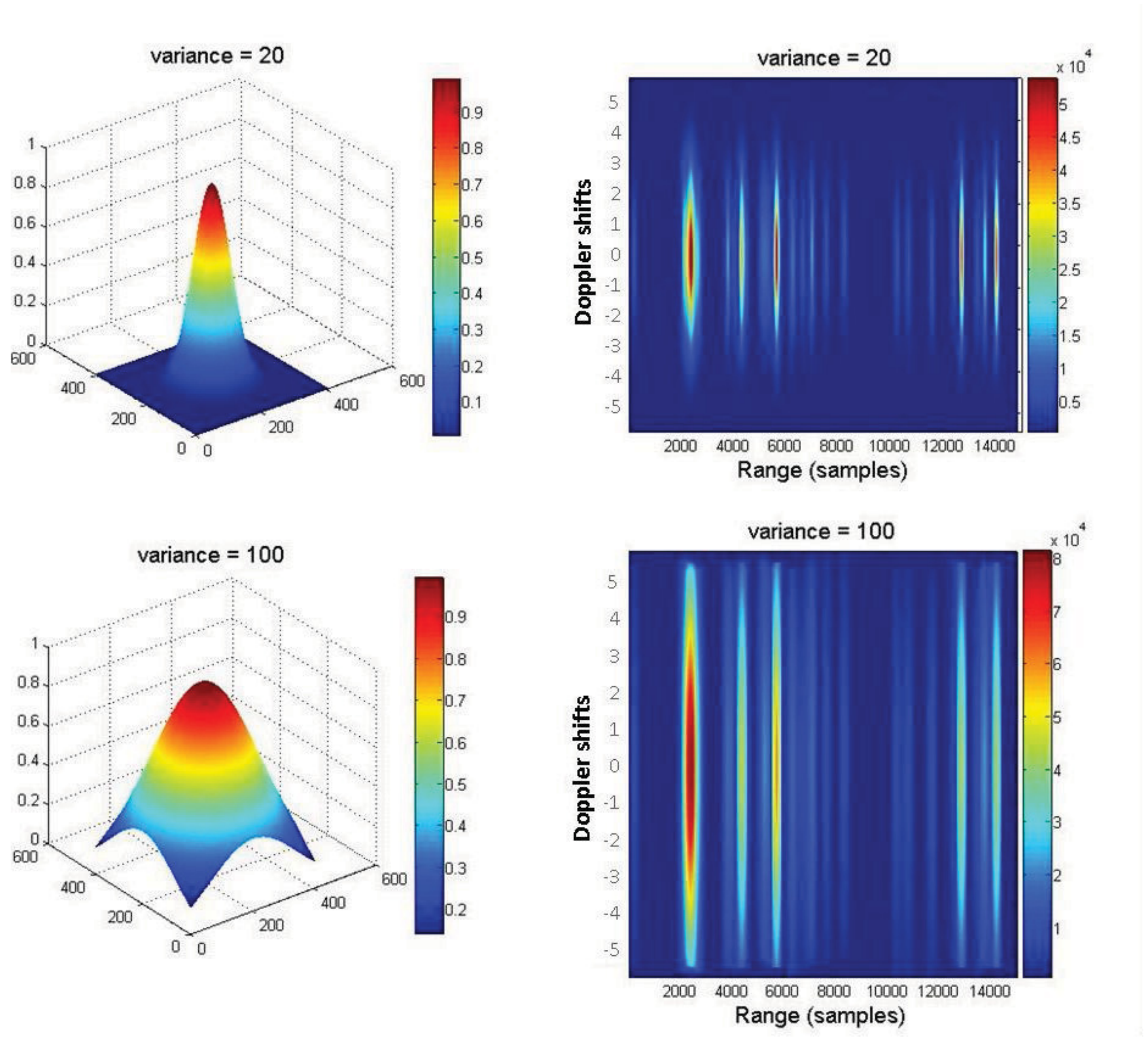

Figure 3.6. Range-Doppler Map Output from Gaussian Convolution Module Using Different Variance Values. The larger variance values result in a wider Gaussian curve, and therefore larger target detection areas in the range-Doppler map.

Adjusting the variance values of the Gaussian curve is equivalent to changing the precision of the collected data. Wider Gaussian curves provide more robust data fusion, but result in less precise target location estimates.

3.3 Data Generation

3.3.1 Signal Generation

The *Signal Generation* module of the simulation code is called first after establishing all user defined parameters. Data is simulated for each microphone array as it would be received directly from each source and echoed from each target in the simulation environment. The data is generated for each sensor individually in each of the microphone arrays. This data is representative of the signal that each microphone would receive based on the type of signal transmitted and the relative phase and distance from the microphone to the target. It is stored in a matrix for each array consisting of a number of rows equal to the number of microphones in the array. The cumulative data matrices represent what a microphone array would produce after receiving a signal in ideal conditions with no noise and no propagation loss.

At this point in the simulation, the simulated data for each source and target is combined into a single matrix for each microphone array. The resulting data for each array is then is padded with zeros to simulate sound delay due to distance from each source or target. The distance calculated for the target is the distance from the source to the target plus the distance from the target to the microphone array. Figure 3.7 below shows an example of zero padded signal data generated by the Signal Generator Module.

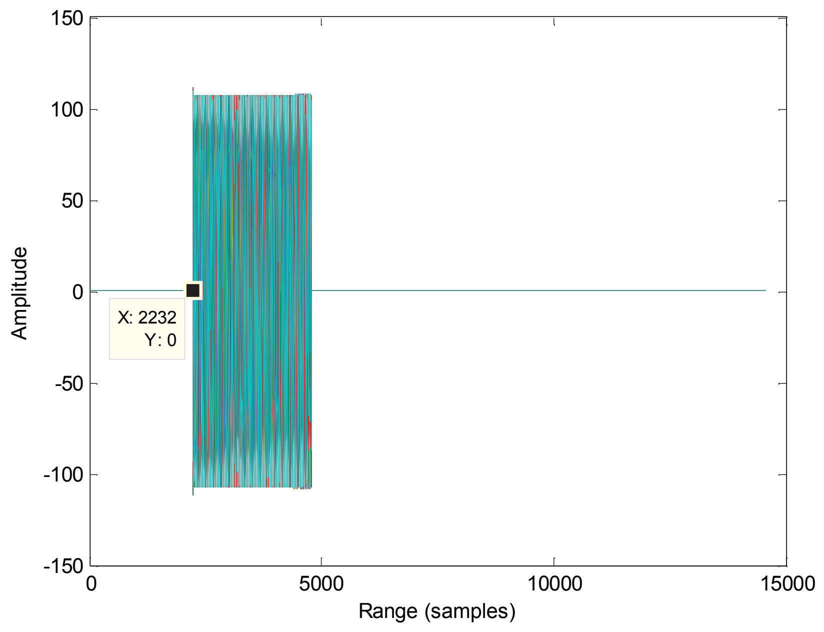

Figure 3.7. Output from the Signal Generator module showing a zero padded signal received by the sensor array. The number of samples before signal in the data is calculated based on the distance from the sound source to the sensor array, including echo distance.

For this example, the sound source was approximately 153 meters away from the sensor array. In this simulation example, the sound waves are assumed to travel through air at 20 degrees Celsius, and so a speed of 343 m/s for the sound waves is assumed. The sound delay in samples can be calculated using the equation 2.6, which results in a sound delay of 2232 samples before the signal begins.

As discussed in the Chapter 2, the orientation of the multi-sensor array causes the individual microphones in the array to be at different linear distances away from the signal source. These result in a small time delay between signal data received at each microphone in the array. The *Signal Generator* captures this phase delay using the user-defined object coordinates. This delay is used later in the *Beamformer* module as the array is steered toward a specific direction. Note that no phase difference can be seen in the data generated for Array 1 because the orientation of Array 1 is perpendicular to the wave path of the signal from the target. This means that theoretically all microphones in a multi-source array will receive the signal simultaneously so long as far field and planar wave assumptions are used. Figure 3.8 shows a detailed view of the output of the Signal Generator module, showing the data for each individual microphone in a 32 sensor array.

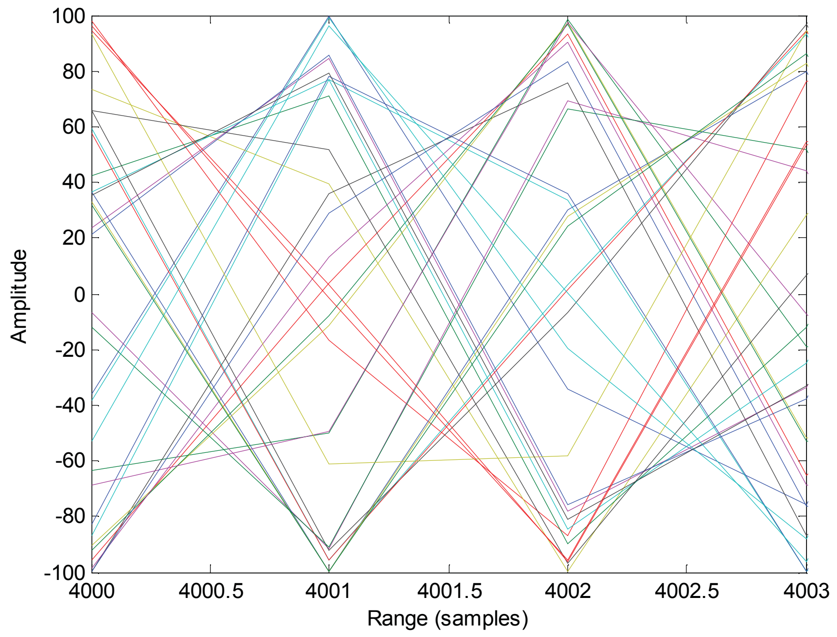

Figure 3.8. Detailed view of the output of the Signal Generator module. This view shows the individual signals from each microphone in the array.

The lines in the figure represent signal data for each sensor in the array. These lines are not aligned due to the difference in time of arrival of the signal for each sensor.

3.3.2 Noise and Propagation Loss Simulator

In order to more accurately represent real world acoustic phenomena, noise and sound propagation loss are simulated. The noise in the received signal can be attributed to environmental sounds as well as electrical noise in the equipment [4]. Propagation loss is primarily attributed to spreading loss and energy absorption. The *Noise and*

Propagation Loss Simulator module of the simulator code alters the data generated by the *Signal Generator* to account for these factors.

Sound propagation loss in real world environments relies on many variables such as the transfer medium, echo surface, ground reflections, wave propagation, and wave interference. For this simulator, not all of these propagation loss factors were considered individually. Calculating the exact intensity loss would require an excessive number of variables to be defined by the user prior to the simulation run. To simplify the calculation, the propagation loss was calculated using only the distance the signal has traveled. This generalization provides adequate propagation loss accuracy for the simulation environment. The propagation loss simulated in this simulation is simply relative to other received signal amplitudes and is directly proportional to the distance from the source to the microphone, including echo distance. The received signal for each generated data matrix is divided by this distance before noise is generated and added.

Noise is simulated by generating a vector of Gaussian distributed, random numbers from negative one to one, equal to the length of the array of data for each microphone. This random integer array is then amplified or reduced according to the desired signal to noise ratio, allowing the user to control the relative amplitude of the applied noise floor. Separate noise matrices are generated for each microphone array and the noise is added to the *Signal Generator* data, simulating additive, Gaussian, white noise.

Figure 3.9 below shows the signal output after the *Noise and Propagation Loss Simulator* module is called. For this example, a Signal to noise ratio of 0.5 was used.

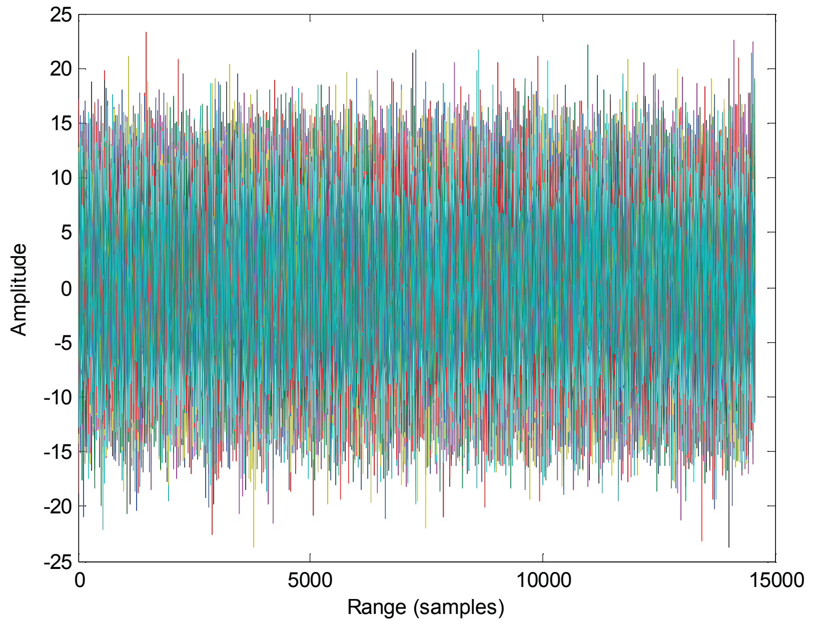

Figure 3.9. Output of Noise and Propagation Loss Simulator module using a signal to noise ratio of 0.5. The original signal is no longer visible at this signal to noise ratio level.

As Figure 3.9 shows, the propagation loss of the signal, and the addition of noise completely obscure the signal by visual inspection. The difficulty in determining the presence of a received signal serves as proof of the need for intelligent data processing for acoustic experimentation.

3.4 Signal Processing

3.4.1 Beamformer

Once the data has been processed by both the *Signal Generator* and the *Noise and Propagation Loss Simulator*, the *Beamformer* module is called to begin processing. The data for each microphone array is beamformed at every angle from -90 degrees to 90 degrees, in one degree increments. The beamformer applies a vector of weight factors to the signals at each channel to steer the array towards the desired angle. These weighted channel signals are summed together, resulting in a single output as opposed to signals for each individual microphone. The output of the beamformer is a vector of data directed towards the given beamforming angle. Figure 3.10 shows the output of the beamformer function for Array 2 at an angle of -45 degrees from the normal plane (directly towards the echo target) with no noise.

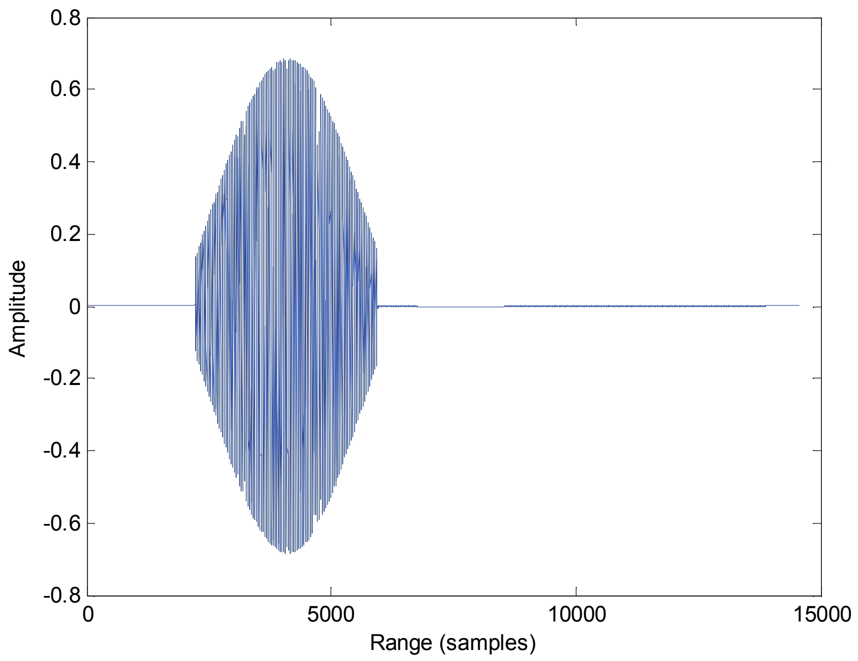

Figure 3.10. Output of the Beamformer module for Array 2, steered directly toward the target at -45 degrees. There is no noise in the system. The output is a single signal as opposed to signals for each individual microphone.

Spectograms of the beamformed data can be seen in Figure 3.11. The first figure shows this data in the frequency-time domain with the addition of noise, the second figure shows the data in the same domain but with noise suppressed.

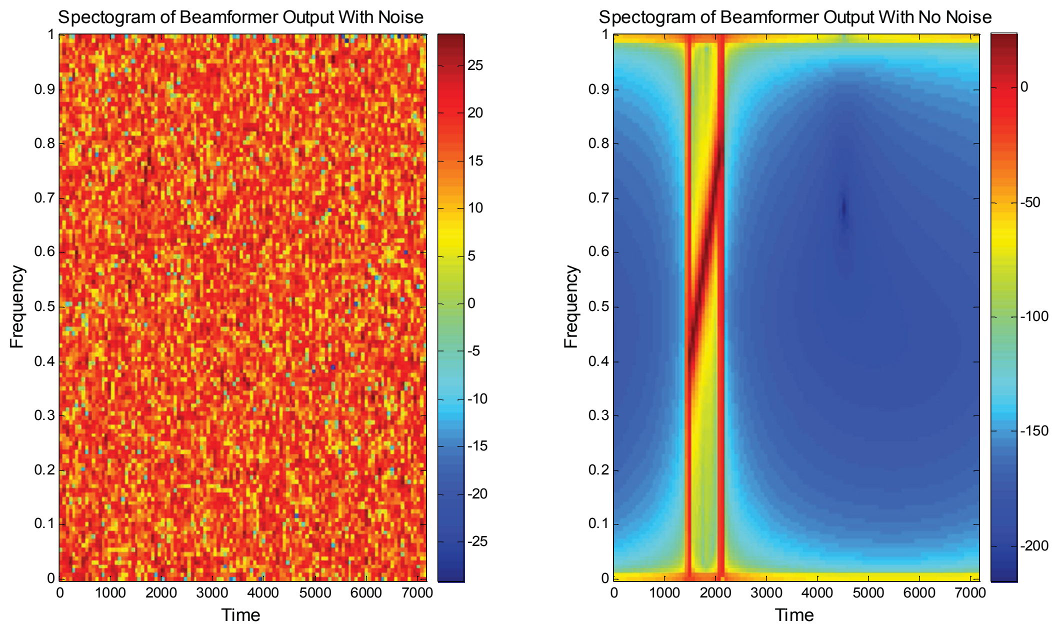

Figure 3.11. Spectogram of the output of the Beamformer module for Array 2 at -45 degrees, The first graph shows the results using an SNR of 0.5, the second graph has all noise suppressed.

While the first figure clearly demonstrates the ambiguity of the detected target location in a noisy environment, the second shows the detected signal at a specific time delay with the sloped chirp frequency characteristic.

3.4.2 Matched Filter

Each *Beamformer* output (beamformed data for each angle for each microphone array), is processed by a matched filter that generates a range-Doppler map for the data. This 2-dimensional map displays the amplitude of the received signal on a range versus

Doppler matrix at a given beamformer angle. A scaled, color gradient image of the range-Doppler map of the Array 2 data at an angle of -45 degrees from the normal plane can be seen in Figure 3.12 with and without noise.

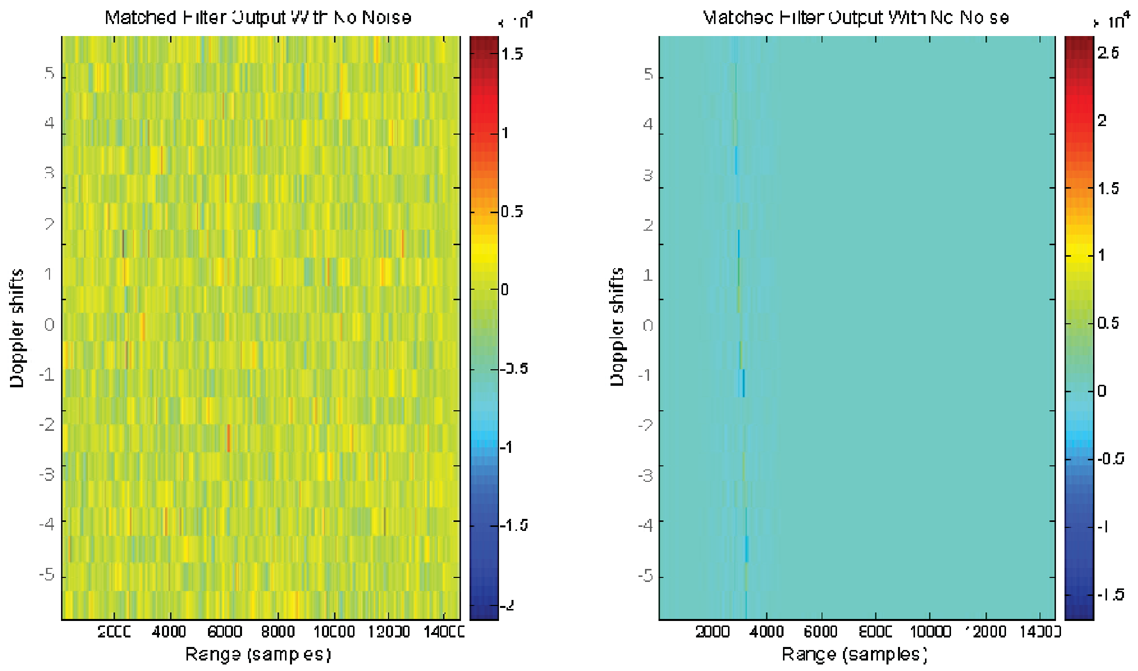

Figure 3.12. Scaled image output of the Matched Filter module for Array 2 at -45 degrees. The first graph show the results of using an SNR value of 0.5, the second graph has all noise suppressed.

A target is detectable in the second figure by visual inspection, though this is more ambiguous in the first figure with noise present. The target in this simulation example is static, and so should have a zero Doppler return. However, some bleeding into other Doppler values is seen due to the chirp signal frequency characteristics. Figure 3.13 shows the cross section of the range-Doppler map at a zero Doppler value.

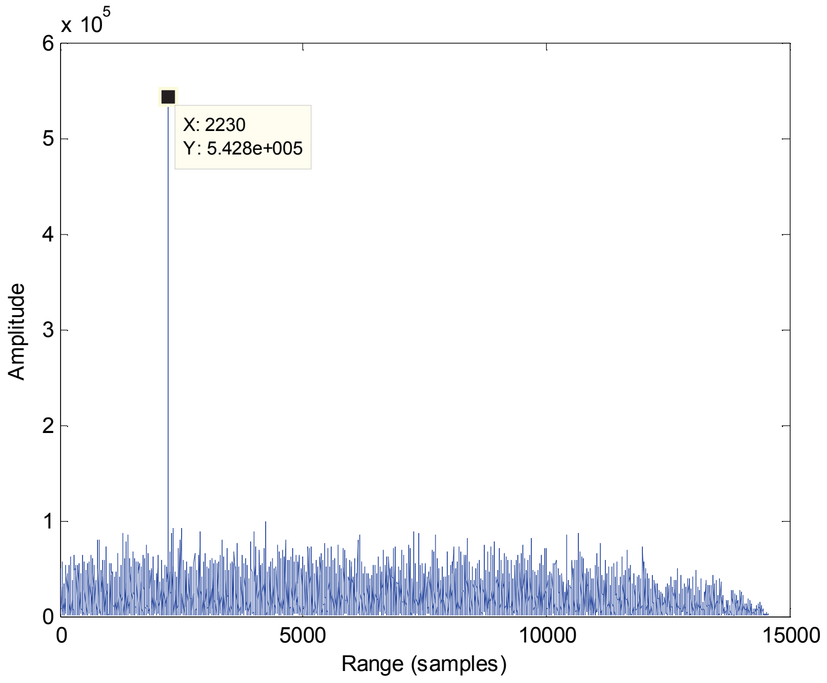

Figure 3.13. Cross section of the range-Doppler map from the Matched Filter module for Array 2 at -45 degrees. This cross section is taken at a neutral Doppler value.

Inspecting the cross section of data at a Doppler value of 11 shows a clear peak return at a range of 2230 samples, which is within two samples or 0.1372 meters of the of the correct target range previously calculated to be 2232 samples. This is an acceptable level of error when considering the size of the area in question is 200 meters by 200 meters.

3.4.3 False Target Generator

Often in real world experimentation, environmental factors attribute to many false positive returns that can obscure the true signal of interest. This is especially true in underwater sonar applications due to the highly reverberating nature of these environments and the large number of sporadic sources such as snapping shrimp [19]. In order to represent this phenomenon, the simulator adds several spurious peaks to the range-Doppler map. These peaks increase the difficulty of accurate target detection, and represent echoes in real world situations. This also helps to demonstrate the effectiveness of the fusion algorithms being analyzed. The peak locations are randomly generated and their amplitude is proportionally larger than that of the signal of interest, to ensure adequate obscurity. Figure 3.14 shows the cross section of data after 20 false positive peak returns are generated.

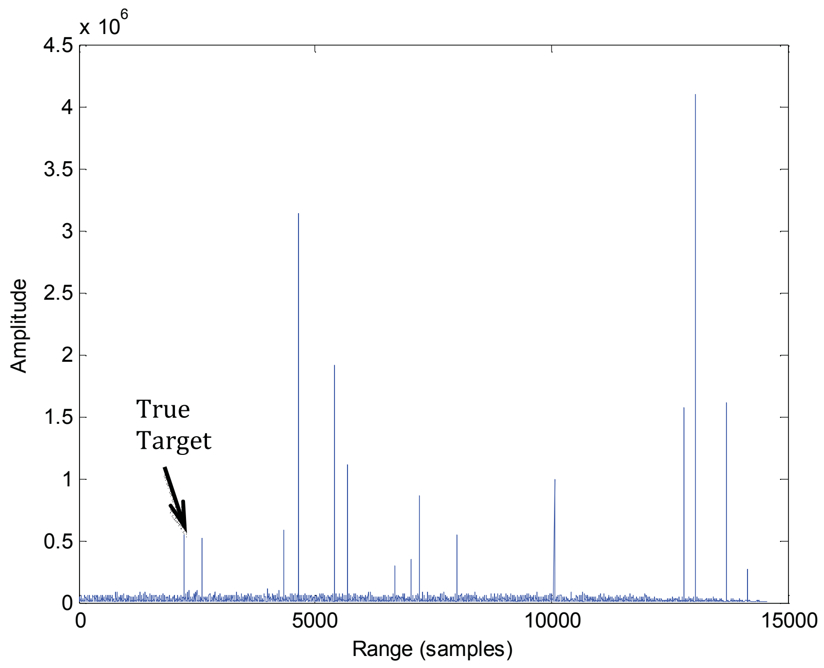

Figure 3.14. Cross section output of the False Target Generator module for Array 2 at -45 degrees. This cross section is from a neutral Doppler value. The figure shows the addition peaks added by the module. The arrow indicates the true target peak.

3.4.4 Impulse Substitution

The next module of the simulator code processes each range-Doppler map individually. When this module is called in the simulator, it steps through each range-Doppler map from the *Matched Filter*, and replaces the peaks with a single impulse. A complex algorithm repeatedly searches through the range-Doppler map data for amplitude values above a certain threshold. The threshold is incrementally reduced until the user-defined number of peaks is found in that search. The location of each peak found above the threshold is then compared to the location of previously found peaks and thrown out if within a specific user-defined range. This ensures that large peaks are not replaced by multiple impulses due to trail-off values of that peak. All values below the minimum peak threshold are reduced to zero. The number of peaks to replace, as well as the minimum distance between peaks are both user-defined values. However, the user should take care when selecting these parameters to ensure that all desired data will be above the minimum threshold. A higher number of peaks to replace and a lower distance between peaks will result in a lower risk of data loss

The following figure shows the effects of the *Impulse Substitution* module. The first graph in Figure 3.15 shows a cross section of a range-Doppler map, showing the signal amplitude along the range axis. The second graph in the figure shows the same cross section after the range-Doppler map has been processed by the *Impulse Substitution* module.

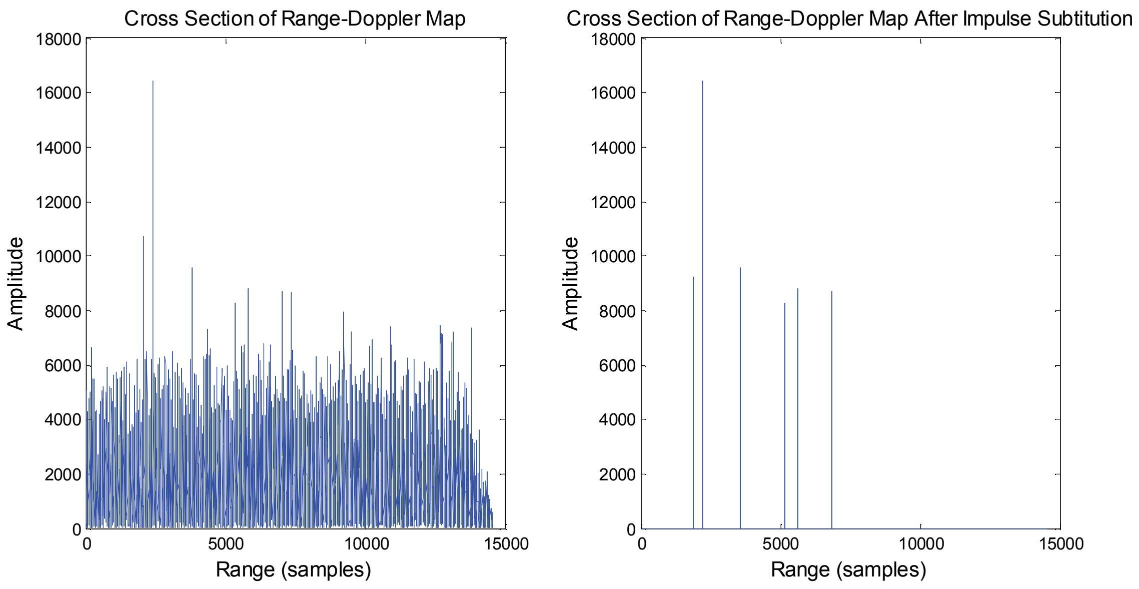

Figure 3.15. Cross section output of a range-Doppler map before and after the Impulse Substitution module for Array 2 at -45 degrees.

3.4.5 Gaussian Convolution

Once the range-Doppler map peaks have been replaced with impulses, a convolution is performed between the modified range-Doppler map and a 2-dimensional Gaussian curve. This step results in a new range-Doppler map with each impulse replaced by a 2-dimensional Gaussian curve centered at the original location of the impulse. Each of these Gaussian curves represents the probability of a target at the given range and Doppler, and the new modified map is called a probability density function. Replacing the impulse map with a probability density function provides robustness to position and orientation uncertainties of the sub-arrays. Target values will be compounded with each other during the fusion process even if the targets are not co-located in different sensor maps. The user-defined variance of the Gaussian curve represents the precision of the data. Figure 3.16 shows the results of the convolution between the impulse substitution output and the Gaussian curve. The range-Doppler map presented is generated from Array 2 for an angle of -45 degrees from the normal plane.

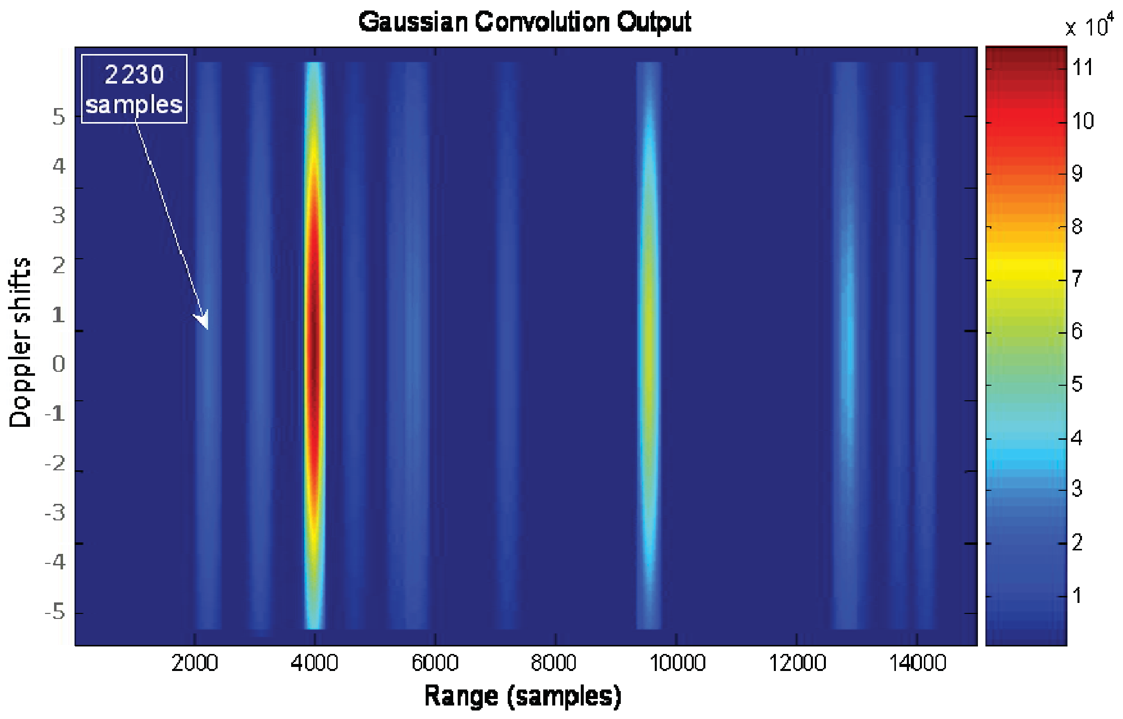

Figure 3.16. Range-Doppler map output of the Gaussian Convolution module for Array 2 at -45 degrees. This figure shows the peak at the true target range of 2230 samples.

Figure 3.17 displays a detailed view of the true target located at 2230 samples. This figure clearly shows the 2-dimensional Gaussian curve characteristic of the target peak.

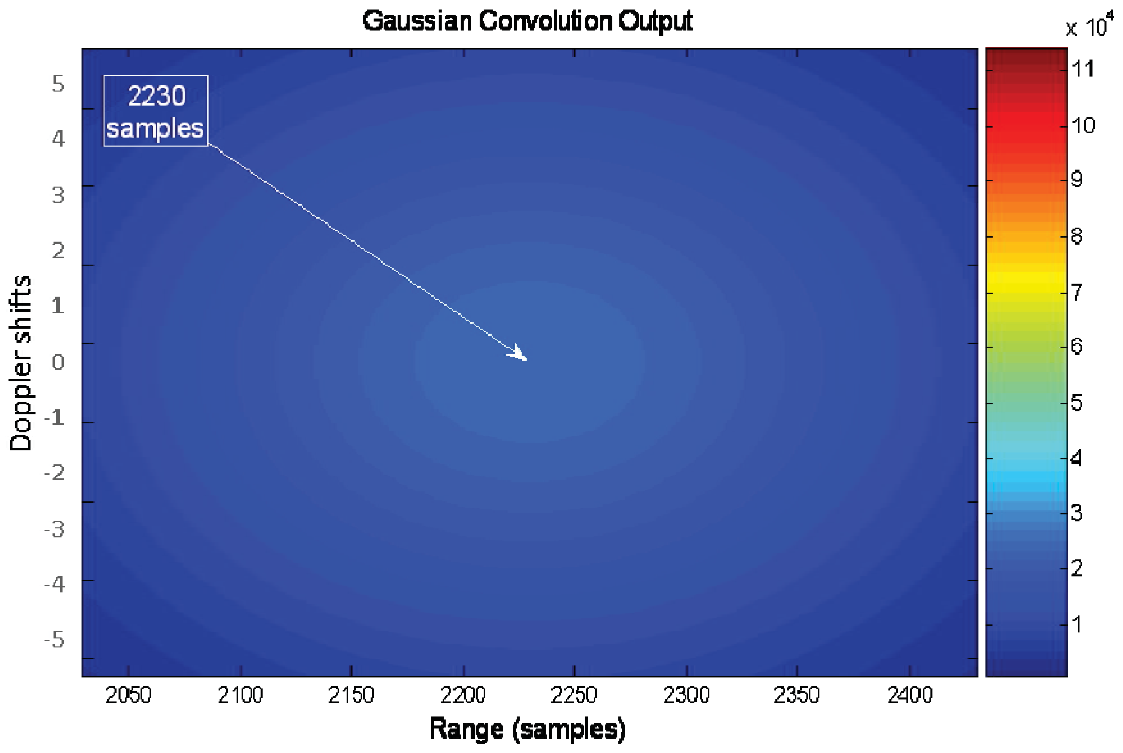

Figure 3.17. Detailed view of the output of the Gaussian Convolution module for Array 2 at -45 degrees. This view shows a close up of the peak at the true target location of 2230 samples.

Changing the variance of the Gaussian curve before convolution is synonymous with changing the uncertainty values of various parameters in the simulation. For example, this uncertainty could be caused from position and orientation errors in the system. A larger Gaussian variance value results in a “wider” cross sectional bell curve, which equates to a wider spread of peak locations. This is advantageous in situation when peaks found from different arrays are determined to be close to each other, but not exactly co-located. The overlapping tail values will still result in a high value after these peaks are multiplied in the fusion process due to the wider bell curve. A larger Gaussian variance can be used to represent less precision in the experimental environment or instrumentation. In the same respect, a smaller variance for the 2-dimensional Gaussian curve results in a “thinner” bell curve and can represent more precise localization results. While a higher variance Gaussian curve results in a more robust fusion process, more iterations are required to achieve a final accurate result.

The number of target peaks displayed is user defined and is generally dependent on the signal to noise ratio and the number of false positive peaks created. Each of these peaks is then processed assuming it is a true target location and all data is retained for fusion.

3.5 Target Processor

3.5.1 Range Correction

The true geometric distance to each of these target peaks must be calculated before their reference point of view can be translated. The distance perceived by the microphone array to the target is not the correct distance between the target and the array, but rather the distance between the target and the array plus the distance between the target and the sound source. The range value for each target in each range-Doppler map must be adjusted to account for this echo distance. The location of the multi-source microphone array and the location of sound source are known, and the total distance from the multi-source array to the target and then to the sound source is measured. The distance from the sound source to the target must be subtracted from this total distance. The figure below illustrates this.

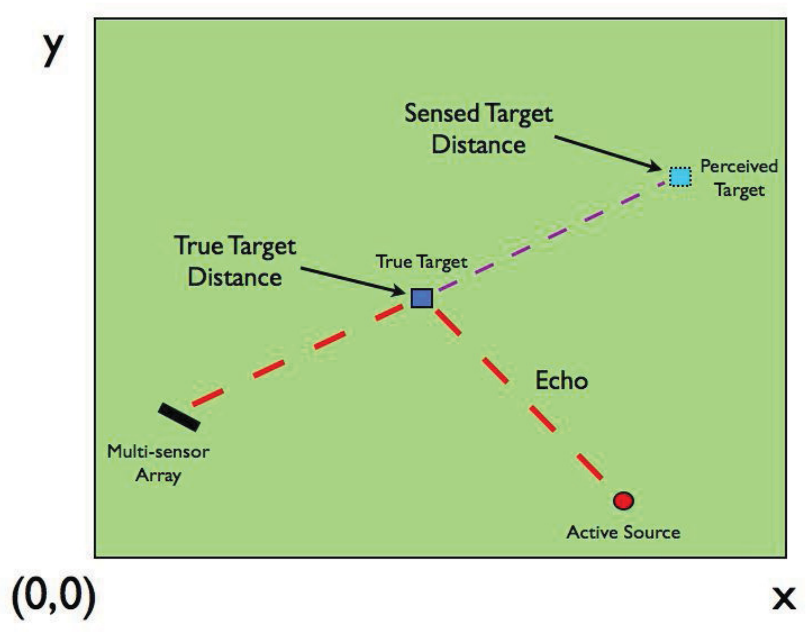

Figure 3.18. A demonstration of perceived target distance due to echo effects. In order to determine the true target distance, the echo distance must be calculated and subtracted.

As Figure 3.18 shows, the sensed distance can be significantly different from the actual target distance due to the echo. The *Range Correction* module calculates and subtracts the echo distance from the sensed target distance to achieve the true target position. It is assumed that because the multi-sensor arrays are networked, they are time-synced and the time the sound signal is sent is known.

This algorithm will not affect the situation where the detected signal is actually the active sound source as the echo distance is calculated to be zero, resulting in an adjusted distance that is equivalent to the original distance.

This geometrically corrected distance is calculated for each target detected at each angle of attack for each multi-sensor array.

3.5.2 Translation

In order for the data from the microphone sub-arrays to be fused, the data from all sub-arrays must be translated to a common frame of reference. The simulator translates all data to the point of view of the origin. The distance and angle with respect to the origin are calculated for each detected target of each range-Doppler map. During the translation process, areas in the map where no target is present, the map amplitude values are set to a baseline value of 0.5. This value ensures that targets not present in all array models are reduced in the multiplication process, but not completely nullified after a single iteration.

Figure 3.19 below shows a new range-Doppler map for an angle of 45 degrees from the origin after each target distance has been calculated and the target peaks have been translated to the proper location with respect to the origin in the map.

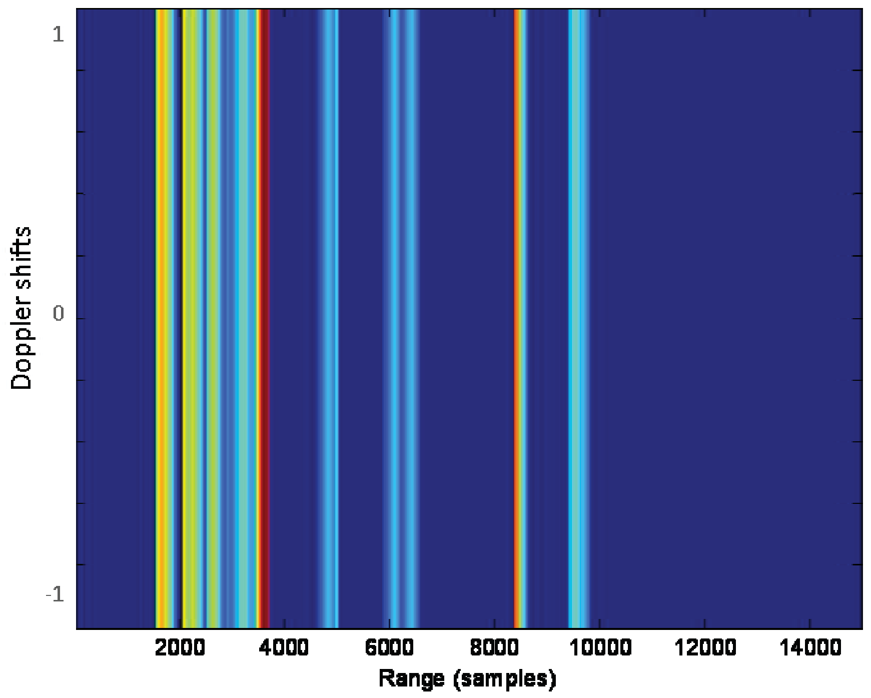

Figure 3.19. Output of the Translation module for Array 2, 45 degrees from the origin. The map is a slice of the whole range-Doppler map at a neutral Doppler value, causing the targets to appear as tall, narrow bands.

In the simulation, the geometrically corrected range-Doppler maps of the arrays are trimmed to just the neutral Doppler value since all targets are assumed static. This is done to reduce system memory usage and simulation run time. The resulting target peaks, as shown in Figure 3.19, appear as narrow bands in the range-Doppler map, rather than the round targets shown in Figure 3.17.

Figure 3.20 shows a cross sectional view of the same map at a zero Doppler value, while Figure 3.21 shows a detailed view of Figure 3.20 locating the true target location.

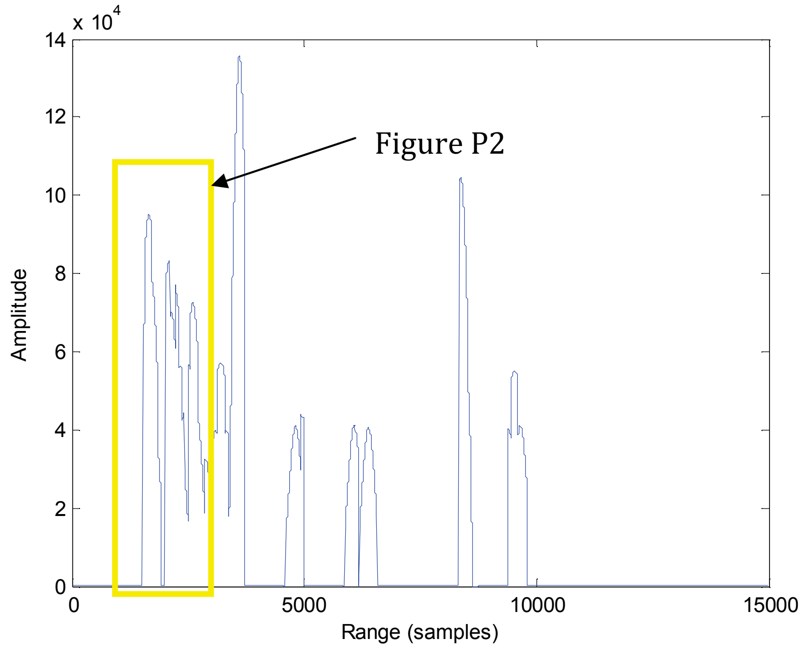

Figure 3.20. Cross sectional view of the output of the Translation module for Array 2 at 45 degrees from the origin. The highlighted box shows the area where the detailed view in Figure 3.21 came from.

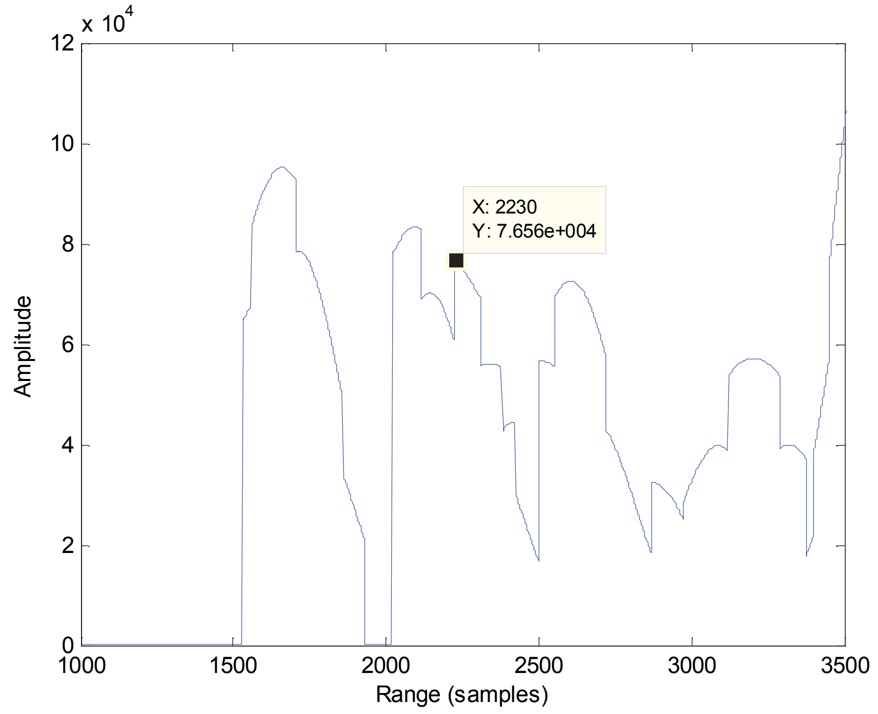

Figure 3.21. Detailed view of the cross section output from the Translation Module for Array 2 at 45 degrees from the origin. The true target location has been marked at 2230 samples.

After geometric correction, there are 91 resulting range-Doppler maps for each multi-sensor array, one for each angle from 0 degrees to 90 degrees with respect to the origin. An angle of 0 degrees coincides with the positive x-axis, and an angle of 90 degrees coincides with the positive y-axis. These range-Doppler maps are stacked together to form a 3-dimensional matrix called the Array Model with axis of Doppler, range, and angle. It is difficult to adequately represent all of this information in a 2-dimensional space in this thesis; however some visual analysis can still be performed.

The figure below shows the maximum value for each range-Doppler map at each angle for array 2.

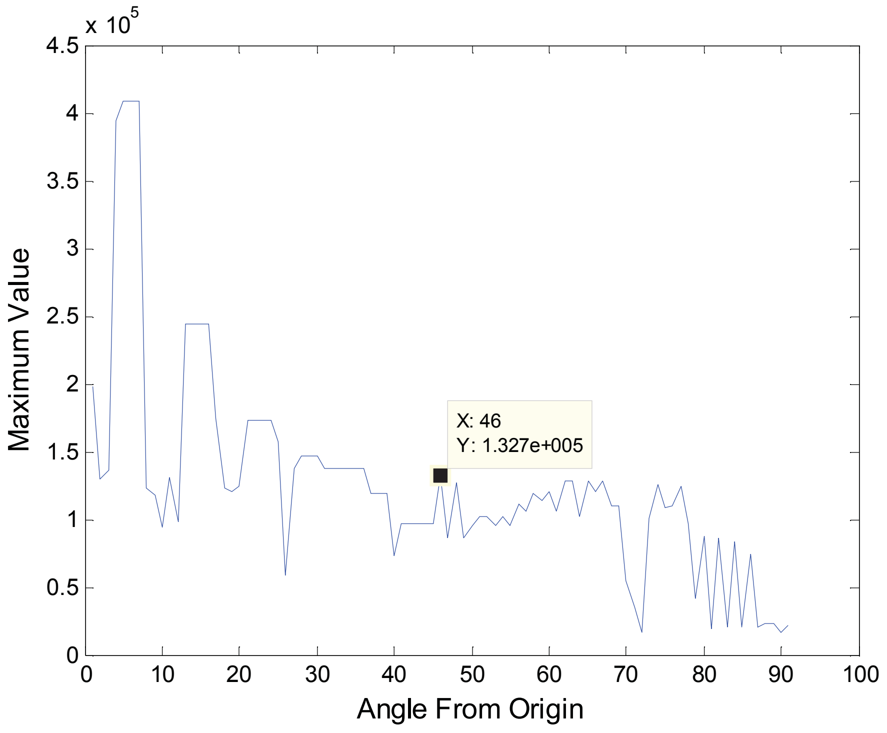

Figure 3.22. Maximum values for each angle from the origin of Array Model 2. The true location at X=46 is highlighted in the figure.

As 3.22 shows, the true target angle cannot be definitively determined because the amplitude of the target location is smaller than other peaks in the array model. Typically in simulation, none of the peaks in this view will represent the true target location, as the amplitude of the true target signal is less than the false target echoes.

3.6 Fusion of Data

3.6.1 Fusion

At this point in the simulator code, a modified range-Doppler probability density map is obtained for each angle step for each microphone array. These maps are stacked together to form one 3-dimensional matrix for each microphone array in an array model. These array models represent the angle from the origin, versus distance from the origin, versus Doppler shift.

The array models are combined in an element by element multiplication process, resulting in a single 3-dimensional matrix referred to as the world model by the simulator. When the array models for each of the arrays are fused, some of the target peak amplitudes are reduced if the peak is not present in all 3 of the array models. This is because the peak values are multiplied by the baseline 0.5 value previously set in the *Translation* module. Similarly, the targets peaks that are present in the same location in all 3 of the world matrices are amplified as they are multiplied together. Figure 3.23 below shows the maximum value for each range-Doppler map at each angle for the world model.

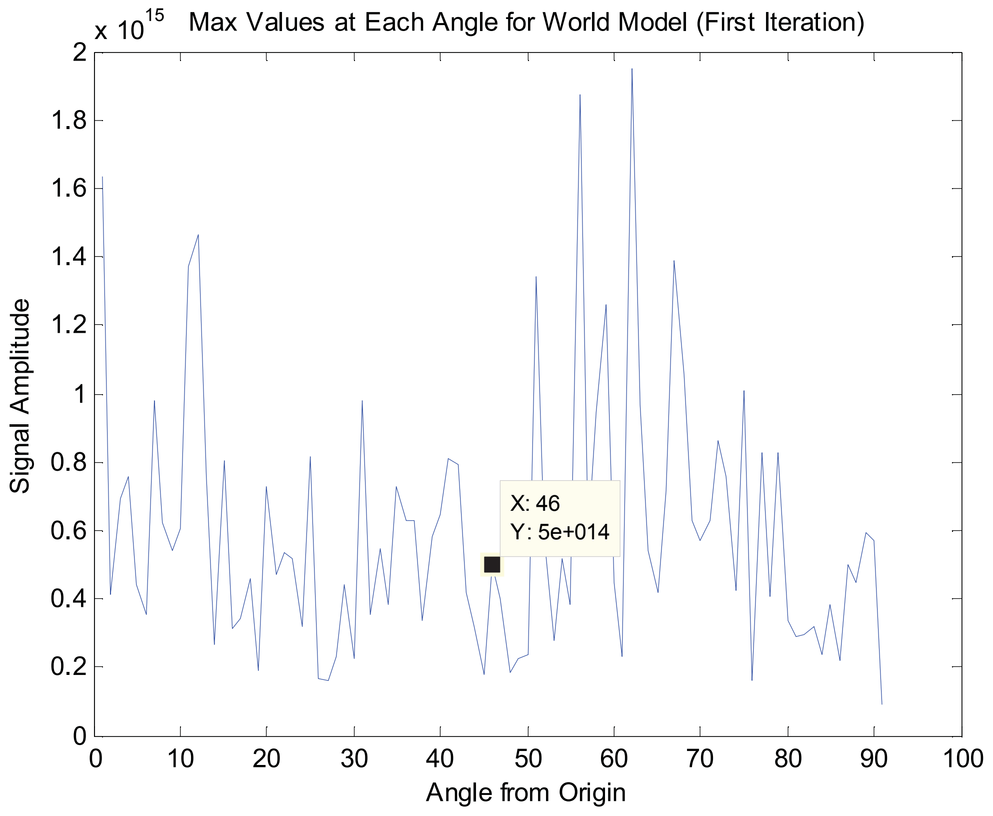

Figure 3.23. Maximum value for each angle from the origin of the World Model after a single iteration. The true target location at X = 46 is shown in the figure.

The true target location is (100m, 100m) and so is located at 45 degrees from the positive x-axis with respect to the origin. The first stack of the World matrix contains the targets located at 0 degrees from the origin, while the second stack contains values located at 1 degree from the origin, etc. Therefore the true target location at 45 degrees should be present in the 46th matrix of the World matrix. As can be seen in Figure 3.23, a peak value is present at the 46th maximum value.

The true target distance from the origin is calculated below using the simulation geometry and equation 2.6:

$$(100m)\sqrt{2} = 141.4m$$

$$d\frac{fs}{c} = 141.4m\frac{5000Hz}{343\,m/s} = 2061 samples$$

A plot of the peak locations at 45 degrees from the origin can be seen in Figure 3.24. The true target peak can be seen at 2061 samples, indicating that the true target is present in the World matrix.

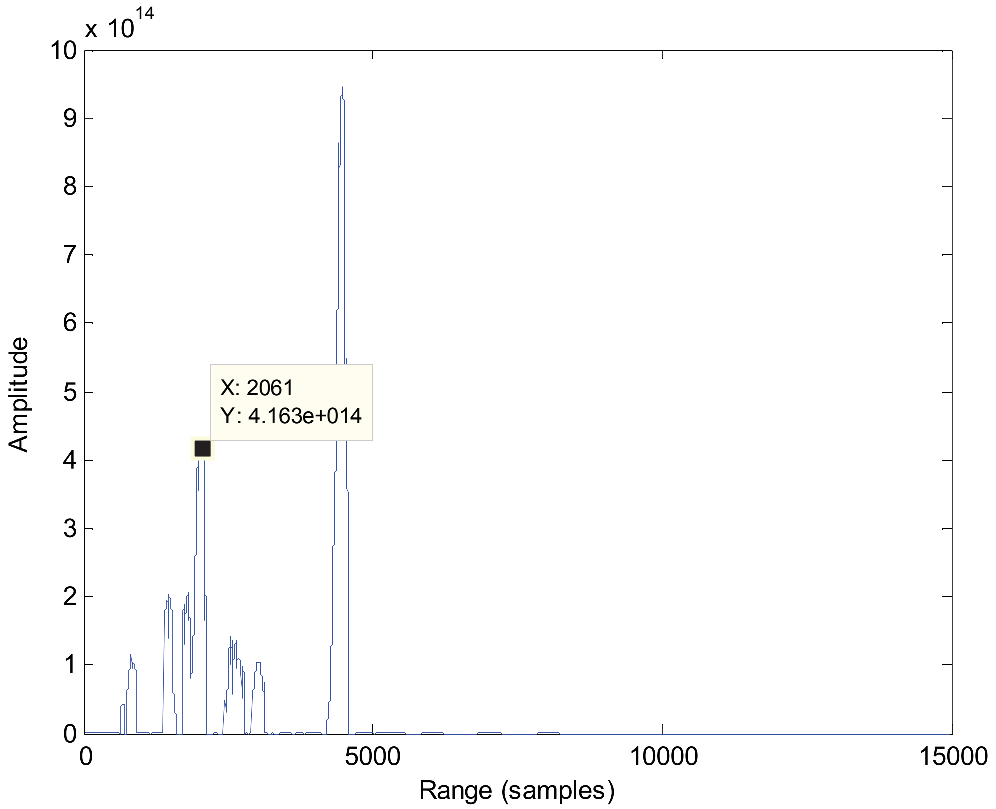

Figure 3.24. World Model at 45 degrees from the origin after one iteration. The true target distance at 2061 samples is shown in the figure.

Though a peak is present at the true target location determining a singular definitive target location would not be possible, as many other potential target locations are also still present in the World matrix. This problem is solved using an iterative process. Experimental data is continually collected and processed into the World model. The iterative process of data collection continually nullifies false positive values while amplifying any common peaks among iterations. Figure 3.25 shows the maximum values at each angle from the origin of the world model after various amounts of iteration.

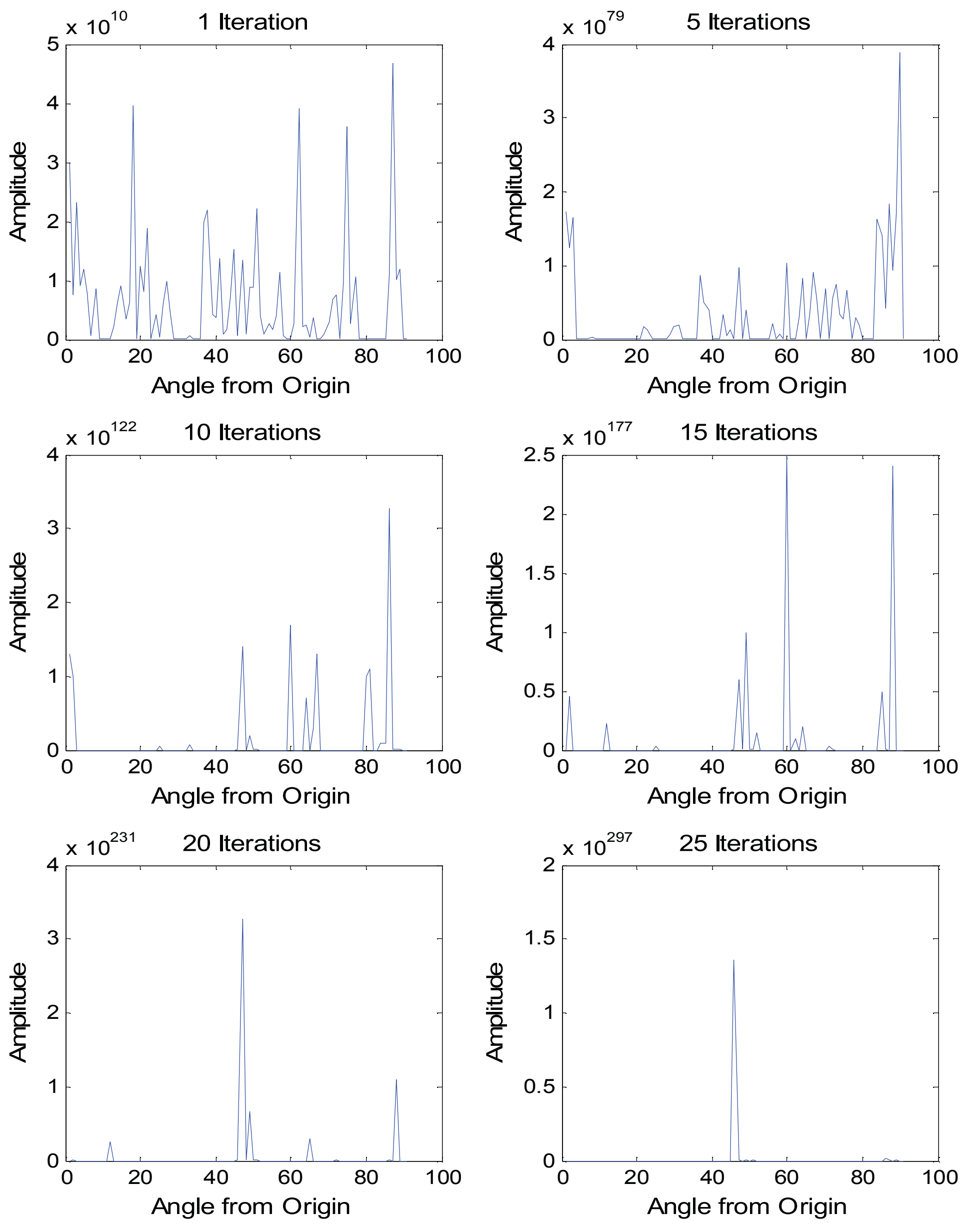

Figure 3.25. Output from the Fusion module after various iterations From top to bottom and left to right, the number of iteration of each figure are 1, 5, 10, 15, 20, and 25. A single, definitive target can only be seen in the final figure after 25 iterations.

It can clearly be seen from Figure 3.25 that a single peak remains in the world model at an angle of 45 degrees from the positive x-axis. Figure 3.26 below shows the range-Doppler map at that 45 degree angle from the positive x-axis.

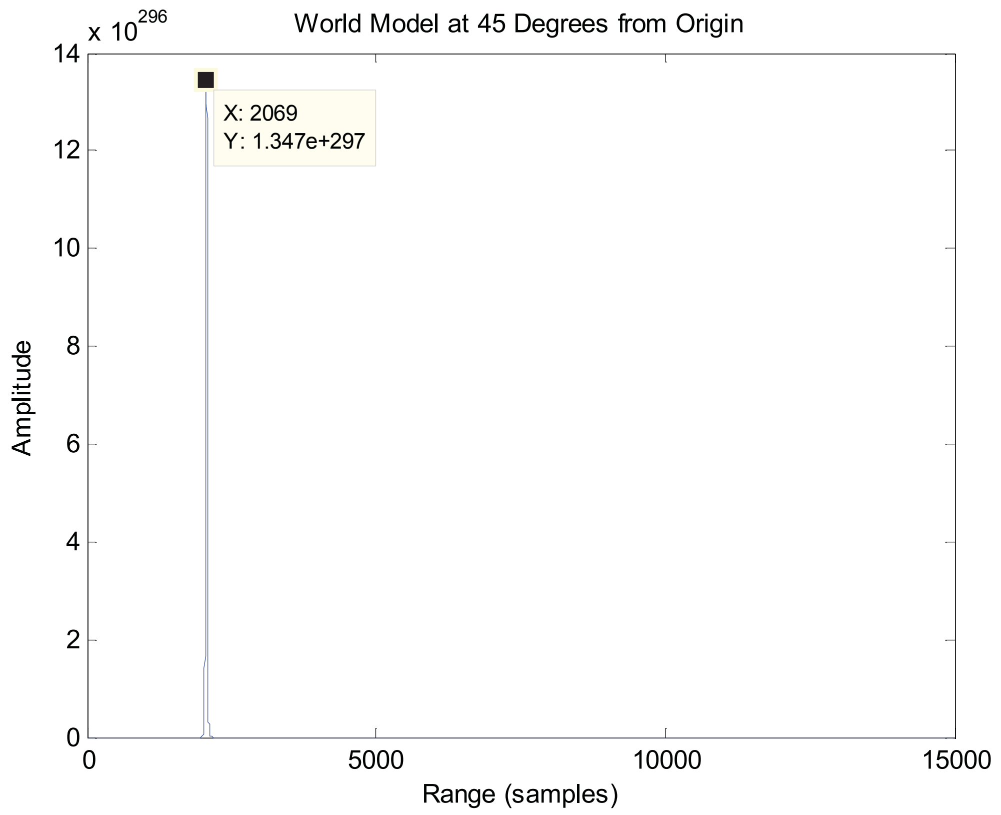

Figure 3.26. World Model at 45 degrees from the origin after 25 iterations. The target is determined to be 2069 samples from the origin, which is 8 samples, or less than 1 meter away from the true target location.

The target locations can clearly be seen at a location of 2069 samples, which equates to 142 meters from the origin. This is within 1 meter from the true target location of 141 meters.

Chapter 4:

Summary and Conclusions

This chapter will review the topics already discussed, as well as offer some final conclusions for this thesis. The focus of this research was to develop a simulation environment for studying networked, multi-sensor microphone arrays using a lossless fusion algorithm model. The simulator serves as a useful tool for comparing different acoustic post-processing algorithms in a controlled, but flexible environment. A thorough overview of the simulator code development was presented, along with an example fusion method and the simulation results.

In chapter 2, background information in acoustic microphone arrays was presented. This review provided a basic understanding of acoustic wavefield properties and propagation. The methods for measuring an objects distance and velocity were discussed with an explanation of active and passive sonar systems. The chapter also included a discussion of beamforming and matched filtering, focusing on the specific the algorithms used in the simulator. Chapter 2 concluded with an overview of current applications and future applications for networked multi-sensor arrays.

Chapter 3 consisted of an in depth explanation of the simulation code development including graphical results for each for each module. After an overview of the simulator architecture, a discussion of the code parameters was presented. These parameters are user defined and their effects on the target detection results were discussed. Each subsequent section of Chapter 3 discussed the specific modules of the simulator in detail. Graphical results of each modules output were presented, and results of the fusion algorithm after various numbers of iterations were also shown. These results demonstrate a successful detection and estimation of the target using the fusion algorithm.

The algorithms presented in this thesis provide a method to accurately simulate active multi-sensor array testing. This thesis demonstrated the effectiveness of these algorithms for detection and estimation across a broad range of acoustic environments and situations. Flexibility and user customization were maintained, ensuring the simulator is a valuable tool for a variety of applications. While further development is needed for handling some more complex acoustic simulations, such as 3-dimensional environments and dynamic targets, a base simulator model for acoustic multi-sensor array testing was achieved.

The Data Generation section of the simulator is able to accurately produce simulated signals received by the microphone arrays based on the input parameters. The noise added to the signal is currently additive, Gaussian, white noise, but further development could provide additional noise types to be simulated. The propagation loss calculation is currently a simplified algorithm based solely on the objects distance from the receiver. This idealized sound propagation loss is used to avoid over complicating the number of user defined parameters required for the Data Generation section.

The Signal Processing section analyses a 180 degree spectrum, in single degree increments, for each multi-sensor array in the simulation. A simple, narrowband, delay and sum beamformer is used in for the *Beamformer* module because the signal types transmitted fit this profile. A different beamformer type could be explored, and would be required if broadband signal types were to be tested. The range-Doppler maps from the *Matched Filter* module have proven to be accurate for all simulated cases tested.

The processing performed by the Target Processing section of the simulator, has also been reliably accurate in testing. The false positives that are generated in the *False Target Generator* module are currently only added to a single Doppler shift value because all targets are assumed to be static. Future versions of this simulator code, if analyzing dynamic targets, could add false targets to the entire range-Doppler map. The *Impulse Replacement* algorithm accurately detects peak values from the range-Doppler map and replaces them with impulses. Currently, careful consideration must be used when choosing the algorithm parameters to ensure that the signal of interest does not fall below the threshold. Code could be developed to determine the required parameters for *Impulse Replacement* to avoid problems with erroneous parameter choices. The *Gaussian Convolution* algorithm could also be developed further. An algorithm to determine the proper variance of the Gaussian curve based upon the received signal could result in more accurate simulations, because the precision of the target location would be determined from actual data characteristics. One limitation to the *Translation* module is that if detected targets are calculated to be near each other after translation, the Gaussian curves could overlap each other in the world map. Currently the most recent target translated will overwrite existing data from previously found targets if overlap occurs. This could result in less accurate results, and future development could merge the two Gaussian curves together more intelligently.

The fusion methods presented are capable of detecting and estimating the correct target of interest from all the collected data after an adequate number of iterations. The current number of iterations required is dependent on the simulated situation. Currently, the number of iterations is considered adequate when a visual examination of the results shows the correct target location as the only peak. Code could be written to determine when adequate iterations have been achieved by comparing the current world model with previous world models to detect changes at each iteration. The true target is assumed found when the location of the target remains constant from one iteration to the next over several iterations. Due to the current fusion method, the amplitudes of the peaks in the world model increase with each iteration as the maps are multiplied together. To improve accuracy, an improved fusion algorithm could be developed to maintain peak amplitude through iterations.

The world model design is a convenient way to package all known data about the simulation environment after processing. However, it is difficult to represent this data visually to the user due to the number of axis in the matrix. An additional program could be used to analyze the final world model and present all the data in a visual method that would facilitate easy studying of the results.

The simulator program presented in this thesis satisfies the research objectives discussed. The flexible, controllable simulation environment produces accurate acoustic representations that can be used in place of, or in support of real world testing. The simulator will be an extremely valuable tool to future researchers in networked multi-sensor acoustic arrays.

Chapter 5:

Future Work

The motivation for the work demonstrated in this thesis was to demonstrate a lossless fusion algorithm using a simulated environment. A acoustic simulator was developed that controlled comparisons to be performed in a flexible environment. The simulator was also intended to serve as a basis for future research in acoustic signal processing. This chapter will discuss future areas of development that can be explored to further the simulator code functionality. Further development of the simulator code will improve the robustness for complex simulated situations, as well as broaden the range of specific algorithms that can be tested.

Future work on the simulator would include developing the ability to process dynamic targets, as the current version of the simulator code will only work for static situations. The *Matched Filter* module estimates the Doppler shift of the target in the creation of the range-Doppler maps, however the target is assumed static and no Doppler translation calculations are performed. Because of this, all example simulations presented in this thesis have a Doppler shift value of zero. The ability to process dynamic targets would greatly broaden the functionality of the simulator, though several changes would have to be addressed. The *Translation* module would need to be able to determine the velocity of the target relative to the origin based on the Doppler shift values in order to perform the necessary translation. Also, the dynamic characteristics of the target would need to be defined by the user as additional parameters. An advanced fusion algorithm would also be required that could handle the drifting of the target location with each simulation iteration.

To increase the realism of the simulator, white, Gaussian noise and a simplified propagation loss model are used to modify the generated signal. Other noise profiles could be added to the system using a more complex algorithm to simulate more specific noise environments when needed. A variable noise could even be applied to the signal that would change over the course of the simulation. Additionally the propagation loss model could be expanded to reflect different sound decay characteristics for more specific simulation testing. For example, simulated testing of acoustics near thermoclines could be performed using more specific sound propagation characteristics.

Currently the simulator environment is limited to 2 dimensions. Extending the capabilities of the software package to handle 3-dimensional environments would greatly increase the usability of the program. A 3-dimensional acoustic simulator would provide a wealth of research opportunities in acoustic multi-sensor array testing. However, adding a third dimension to the testing environment would significantly increase the complexity of the algorithms used. The Array Models and World Models would require an addition dimension to hold the data, and the geometric calculations performed in the simulator would increase in complexity. A more complex sound propagation behavior would be required to realistically simulate sound in a 3-dimensional space. Additionally, the simulator processing time would increase significantly per iteration, as the amount of data to be processed would greatly increase.

To increase the flexibility of the simulator, some of the current system constants could be changed to user-definable parameters, offering more control over the simulation environment. This would require additional algorithm development would be required to handle the extra parameters. Currently the simulator assumes an in-air testing environment in ideal conditions. Allowing the user to specify alternate propagation mediums under various conditions would tailor the simulator to more specific acoustic research. The simulator could be adapter to handle other wavefield types, such as vibrational or electromagnetic, allowing hybrid wavefield simulations to be tested. These additional user-defined parameters directly increase the usefulness and robustness of the simulator program.

References

[1] Benesty, J., Chen, J., Huang Y. “Microphone Array Signal Processing. Springer Topics in Signal Processing”, v. 1. Berlin: Springer, 2008.

[2] Bell, K., “MAP-PF Tracking with a Network of Sensor Arrays,” Proceedings of the Acoustic, Speech and Signal Processing, Vol. 4, 2005

[3] Chen J., Yao, K., Hudson, R. “Acoustics Source Localization and Beamforming: Theory and Practice,” *EURASIP Journal on Applied Signal Processing*, vol. 4, 2003.

[4] Davenport, W., Root, W. An Introduction to the Theory of Random Signals and Noise. New York: IEEE Press, 1987.

[5] Erling, J., Roan, M., Gramann, M. “Performance Bounds for Multisource Parameter Estimation using a Multiarray Network,”

[6] Gold, B. “Performance Analysis of Active and Passive Multi-Array Sonar Networks”

[7] Hoppe, E. “Improving Signal Clarity through Interference Suppression and Emergent Signal Detection”

[8] Kagami, S., Mizoguchi, H,. Tamai, Y. “Microphone Array for 2D Sound Localization and Capture,” *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 04*, vol.1, 2004.

[9] Kinsler, L.E., Frey A.R., Coppens, A.B., Sanders, J.V. Fundamentals of Acoustics, 4th Edition. John Wiley and Sons, Inc. 2000.

[10] Leivs, P., Gay, D., Culler, D. “Active Sensor Networks,” Fuzzy Information Processing Society, 1997.

[11] Li, D., Wong, K., Hu, Y., Sayeed, A. “Detection, Classification and Tracking of Targets in Distributed Sensor Networks,” IEEE Signal Processing Magazine, Vol. 19, No. 2, March 2002

[12] Li, J., Stoica, P. Robust Adaptive Beamforming Hoboken, NJ: John Wiley, 2006.

[13] McErlean, D., Narayanan, S. “Distributed Detection and Tracking in Sensor Networks,” Proceedings, 36th Asilomar Conf. Signals, Systems & Computers, Pacific Grove, CA, 2002

[14] Pierce, A. D. Acoustics: An Introduction to Its Physical Principles and Applications. Acoustical Society of America. 1989.

[15] Poor, H. V., “An Introduction to Signal Detection and Estimation. Springer Texts in Electrical Engineering”, New York: Springer-Verlag, 1994.

[16] Potamitis, I., Huimin, C., Tremoulis, G. “Tracking of Multiple Moving Speakers with Multiple Microphone Arrays,” IEEE Trans. on Speech and Audio Processing, Vol. 12, Sept. 2004

[17] Sibul, L., Roan, M., Schwartz, S., Coviello, C. "Lossless Information Fusion for Active Ranging and Detection Systems," IEEE Transactions on Signal Processing, Vol. 54, No. 10, Oct. 2006

[18] Stojanovic, M. "Recent Advances in High-Speed Underwater Acoustic Communications." IEEE Journal of Oceanic Engineering, Vol. 21. April 1996.

[19] Urick, R.J. Principles of Underwater Sound, 3rd Edition. Peninsula Publishing. Los Altos, CA. 1983.

[20] Valenti, M. "An Optimal Soft-Output Multiuser Detection Algorithm and its Applications"

[21] Van Trees, H., Optimum Array Processing, Detection, Estimation, and Modulation Theory, Part IV. New York: Wiley, 2002.

[22] Wang Z., Li J., and Wu R. "Time-Delay- and Time-Reversal-Based Robust Capon Beamformers for Ultrasound Imaging," *IEEE Transcript on Medical Imaging*, vol. 24, no. 10.

Appendix A:

Source Code

A.1 VAL_simulator_main.m

```
tic

for iter = 1:1

warning off

clear aDataZ aDataZn arrayDataS arrayDataT bformT bformTtest doppEnd
clear doppPeak doppSize doppStart newout newoutF newoutT newoutTtest
clear newouttest

clear nnoise peaks rangeEnd rangePeak rangeSize rangeStart sig t temp
clear tempsize world1 world2 world3

load falseTargLocs.mat

%workspace (0,0) at the bottom left corner

j = 0
numArrays = 3;

%tVars and sVars elements are x and y coords
tVars = [100,100];
sVars = [80,20];

%arrayVars elements = num of elements, x coord, y coord, orientation
arrayVars(1,:) = [32,10,10,45/180*pi,0,51.3/180*pi];
arrayVars(2,:) = [32,50,150,270/180*pi,-45/180*pi,18.4/180*pi];
arrayVars(3,:) = [32,180,80,180/180*pi,14/180*pi,-45/180*pi];
elemSpace = 0.025;
c = 343;

%G is the Gaussian curve that replaces the peak in newout
xg = 0:.1:40; yg = 0:.1:40; xgo = 20; ygo = 20;
for c1 = 1:length(xg)
   for c2 = 1:length(yg)
      G(c1,c2) = exp(-((((xg(c1)-xgo).^2)/(2*200))+(((yg(c2)-ygo).^2)/(2*200))));
   end
```

```
end

fs = 5000;
cLength = 0.25;
t = 0:1/fs:cLength;
cMin = 1000;
cMax = 2000;
sig = 100*chirp(t,cMin,cLength,cMax);
%sig = 100*rand(1,100);
%sig = 100*sin(2*pi*t*1500);
noiseBool = 1; %use 1 for noise, use 0 for no noise
snr = 2; %noise variable
cali_tau = zeros(1,arrayVars(1,1)); %calibration constants
cali_amps = ones(1,arrayVars(1,1)); %calibration constants

for j = 1:numArrays
   j
   %determine angle from array to target and from array to source
   [at(j)] = angle_from_array(arrayVars(j,2:3),arrayVars(j,4),tVars(1:2));
   [as(j)] = angle_from_array(arrayVars(j,2:3),arrayVars(j,4),sVars(1:2));

   %determine distance from array to target and from array to source
   dSourceTarg = sqrt((tVars(2) - sVars(2))^2 + (tVars(1) - sVars(1))^2);
   dt(j) = sqrt((tVars(2) - arrayVars(j,3))^2 + (tVars(1) - arrayVars(j,2))^2) +
dSourceTarg;
   ds(j) = sqrt((sVars(2) - arrayVars(j,3))^2 + (sVars(1) - arrayVars(j,2))^2);

   %distance in terms of samples
   dts(j) = dt(j)/343*fs;
   dss(j) = ds(j)/343*fs;

   %generate mic array data from source and target and simulate decay
   arrayDataT{:,:,j} = Data_Ang_Gen(sig,fs,at(j),5,arrayVars(j,1),elemSpace,c);
   arrayDataT{:,:,j} = arrayDataT{:,:,j}./(dt(j));
   arrayDataS{:,:,j} = Data_Ang_Gen(sig,fs,as(j),5,arrayVars(j,1),elemSpace,c);
   arrayDataS{:,:,j} = arrayDataS{:,:,j}./(ds(j));
   sizeDataT = size(arrayDataT{:,:,j});
   sizeDataS = size(arrayDataS{:,:,j});

   %pad with zeroes to simulate distance and add noise
   aDataZ{:,:,j} = zeros(arrayVars(j,1), 1000/c*fs);
   tempDataZ = aDataZ{:,:,j};
   tempDataZ(:,dts(j):(dts(j)+sizeDataT(2)-1)) = arrayDataT{:,:,j};
   %tempDataZ(:,dss(j):(dss(j)+sizeDataS(2)-1)) = arrayDataS{:,:,j};
   aDataZ{:,:,j} = tempDataZ;
   clear tempDataZ;
```

```
    if noiseBool == 1;
        aDataZn{:,:,j} = awgn(aDataZ{:,:,j}, snr);
    else
        aDataZn{:,:,j} = aDataZ{:,:,j};
    end

%   find target angles
%   bAngs = [-60, 5, 60];
%   beamformer(:,:,j) = bform_v5(aDataZn{:,:,j}, bAngs, elemSpace, c, fs,
(cMin+cMax)/2, (cMin+cMax)/2, arrayVars(j,1), cali_tau,cali_amps);

%   peakAngs = peakfinder2D(beamformer(:,:,j), 5, 3, 0.5);
    angStep = 1; %be sure to change angle calc if you change this
    numAngs = 180/angStep;
    bformTtest = bform_ang3(aDataZn{:,:,j}, 30,
elemSpace,c,(cMin+cMax)/2,arrayVars(j,1),cali_tau,cali_amps);
    newouttest = wave(bformTtest, sig, 0.9, 1.1, 0.01, 1, 0, 0); %only calculated to
determine size of newoutT
    temp = spikemaker(newouttest, 3, 20, .5, G, c, fs);
    tempsize = size(temp);
    newoutTtest = temp(round(tempsize(1)/2)-5:round(tempsize(1)/2)+5,:,:);
    clear temp;

    [r, rr, rrr, rrrr] = size(newoutTtest);

    if j == 1
        world1 = ones(r, rr, 91);
        world1 = world1./5;
    end

    if j == 2
        world2 = ones(r, rr, 91);
        world2 = world2./5;
    end

    if j == 3
        world3 = ones(r, rr, 91);
        world3 = world3./5;
    end

    for ang = 1:numAngs
        angle = -91 + (angStep*(ang));

        %beamform data at desired angles
        bformT = bform_ang3(aDataZn{:,:,j}, angle,
elemSpace,c,(cMin+cMax)/2,arrayVars(j,1),cali_tau,cali_amps);

        %generate range-Doppler map of beamformed data
        newout = wave(bformT, sig, 0.9, 1.1, 0.01, 1, 0, 0);

        %simulate false targets
        newoutF = newout;
        trueTargAmp = max(newout(11,:));
        [E,R] = size(newoutF);

        for iii = 1:20
          falseLoc = falseTargLocs(ceil(rand*length(falseTargLocs)));
          falseAmp = 3*randn*trueTargAmp;
          newoutF(11, falseLoc) = falseAmp;
        end

        %this replaces the range-Doppler map peaks with Gaussian curves
        temp = spikemaker(newoutF, 35, 2, .1, G, c, fs);
        %tempsize = size(temp);
        newoutT = temp(9:13,:,:);
        %newoutT = temp;

        %geometry calculations to figure out true object location
        [peaks] = peakfinder3D(newoutT, 30, 20, 0.1);
        doppPeak  = peaks(:,1);
        rangePeak = peaks(:,2);

        for u = 1:length(doppPeak)
          XY = rangePeak(u)/fs*c;
          angA = abs((arrayVars(j,4)-as(j)*pi/180) - (arrayVars(j,4)-(angle*pi/180)));
          W = sqrt(XY^2 + ds(j)^2 - 2*XY*ds(j)*cos(angA));
          angB = asin(ds(j)*sin(angA)/W);
          angC = pi - 2*angB;
          Ydist = W*sin(angB)/sin(angC);
          targDist = XY - Ydist; %actual distance between array and object
          if as(j) >= angle-angStep/2 && as(j) <= angle+angStep/2
            targDist = XY;
          end

          newAngt = arrayVars(j,4) - (angle*pi/180);

          if newAngt > 2*pi
            newAngt = newAngt - 2*pi;
          end
```

```
        if newAngt < 0
            newAngt = newAngt + 2*pi;
        end
        if newAngt < pi/2
            yCoord = arrayVars(j,3) + targDist*sin(newAngt);
            xCoord = arrayVars(j,2) + targDist*cos(newAngt);
        elseif newAngt < pi
            newAngt = pi - newAngt;
            yCoord = arrayVars(j,3) + targDist*sin(newAngt);
            xCoord = arrayVars(j,2) - targDist*cos(newAngt);
        elseif newAngt < 3*pi/2
            newAngt = newAngt - pi;
            yCoord = arrayVars(j,3) - targDist*sin(newAngt);
            xCoord = arrayVars(j,2) - targDist*cos(newAngt);
        else
            newAngt = 2*pi - newAngt;
            yCoord = arrayVars(j,3) - targDist*sin(newAngt);
            xCoord = arrayVars(j,2) + targDist*cos(newAngt);
        end

%xCoord and yCoord are x and y coordinates of the object with respect to origin

        if yCoord >= 0 && xCoord >= 0
            targDFO = sqrt(yCoord^2 + xCoord^2); %object distance from origin
            targAFO = atan(yCoord/xCoord)*180/pi; %object angle from origin

            doppSize = 4;
            rangeSize = 200;
            doppEnd = doppPeak(u)+doppSize;
            doppStart = doppPeak(u)-doppSize;
            rangeEnd = rangePeak(u)+rangeSize;
            rangeStart = rangePeak(u)-rangeSize;
            rangeSize = [rangeSize,rangeSize];
            [d_map,r_map] = size(newoutT);

            if doppEnd > d_map
                doppEnd = d_map;
            end

            if doppStart < 1
                doppStart = 1;
            end

            if rangeEnd > r_map
                rangeEnd = r_map;
                rangeSize(2) = r_map - rangePeak(u);
            elseif rangeStart < 1
               rangeStart = 1;

               rangeSize(1) = rangePeak(u) - 1;
            end

            if round(targDFO*fs/c) - rangeSize(1) <= 0
               rangeSize(1) = round(targDFO*fs/c) - 1;
               rangeStart = rangePeak(u) - (round(targDFO*fs/c)-1);
            end

            if round(targDFO*fs/c) + rangeSize(2) > r_map
               rangeSize(2) = r_map - round(targDFO*fs/c);
               rangeEnd = rangePeak(u) + (r_map - round(targDFO*fs/c));
            end

            if j == 1
               if xCoord > tVars(1)-7 && xCoord < tVars(1)+7 && yCoord >   tVars(2)-7
&& yCoord < tVars(2)+7
                  world1(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
               elseif xCoord > sVars(1)-7 && xCoord < sVars(1)+7 && yCoord >
sVars(2)-7 && yCoord < sVars(2)+7
                  world1(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
               else
                  world1(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd);
               end
            end

            if j == 2
               if xCoord > tVars(1)-7 && xCoord < tVars(1)+7 && yCoord > tVars(2)-7
&& yCoord < tVars(2)+7
                  world2(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
               elseif xCoord > sVars(1)-7 && xCoord < sVars(1)+7 && yCoord >
sVars(2)-7 && yCoord < sVars(2)+7
                  world2(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
               else
```

```
                    world2(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd);
                end
            end

            if j == 3
                if xCoord > tVars(1)-7 && xCoord < tVars(1)+7 && yCoord > tVars(2)-7
&& yCoord < tVars(2)+7
                    world3(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
                elseif xCoord > sVars(1)-7 && xCoord < sVars(1)+7 && yCoord >
sVars(2)-7 && yCoord < sVars(2)+7
                    world3(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd)*1;
                else
                    world3(doppStart:doppEnd, round(targDFO*fs/c)-
rangeSize(1):round(targDFO*fs/c)+rangeSize(2), round(targAFO)+1) =
newoutT(doppStart:doppEnd, rangeStart:rangeEnd);
                end
            end
        end
    end

%     if angle == 0 & j == 1
%         figure, imagesc(newoutT)
%         test1 = [XY, Ydist, targDist, newAngt, xCoord, yCoord, angA*180/pi,
angB*180/pi, angC*180/pi, W, targDFO, targAFO];
%     elseif angle == -45 & j == 2
%         figure, imagesc(newoutT)
%         test2 = [XY, Ydist, targDist, newAngt, xCoord, yCoord, angA*180/pi,
angB*180/pi, angC*180/pi, W, targDFO, targAFO];
%     elseif angle == 15 & j == 3
%         figure, imagesc(newoutT)
%         test3 = [XY, Ydist, targDist, newAngt, xCoord, yCoord, angA*180/pi,
angB*180/pi, angC*180/pi, W, targDFO, targAFO];
%     end
  end
end

if (length(world1) > 14977)
  world1 = world1(:,1:14977,:);
end
```

```
if (length(world2) > 14977)
   world2 = world2(:,1:14977,:);
end
if (length(world3) > 14977)
   world3 = world3(:,1:14977,:);
end

multString = sprintf('WORLD%d = world1.*world2.*world3;',iter);
eval(multString);
varString = sprintf('WORLD%d', iter);
saveString = sprintf('WORLDTESTFILE%d.mat', iter);
save(saveString, varString);
clearVar = sprintf('clear WORLD%d', iter);
eval(clearVar);
end

toc

%WORLDTEST = WORLD.^10;
%WORLD3 = WORLD;
%save('world3.mat','WORLD3');
```

A.2 angle_from_array.m

```
function [thetadeg] = angle_from_array(arraypoint,arrayangle,targetpoint)

a = [targetpoint(1)-arraypoint(1),targetpoint(2)-arraypoint(2),0];
b = [cos(arrayangle),sin(arrayangle),0];

theta = mod(atan2(a(1)*b(2)-b(1)*a(2),a(1)*b(1)+a(2)*b(2)),pi*2);
thetadeg = theta*180/pi;

if thetadeg > 180
   thetadeg = thetadeg - 360;
else
   thetadeg;
end
```

A.3 Data_Ang_Gen.m

```
function [data] = Data_Ang_Gen(y1,fs,ang,op,n_elem,elem_d,c)

%y1 = ping signal
%fs = sampling freq
%ang = angle to target (zero is straight ahead, right pos)
%op = how many times do you want to upsample (suggested sample freq of 50k,
%   upsample of 4)
%n_elem = number of elements in array
%elem_d = spacing between elements (in meters)
%c = speed of sound (m/s)
%will have to calculate the angle for each array to target. Will also have
%to zero-pad the output data for each array to simulate range. Will run
%program for each array.

y11 = resample(y1,op,1);
ang1=abs(ang);
tau1 = (elem_d*sind(ang1)/c)*(0:(n_elem-1));
samp1 = abs(round(tau1*(fs*op)))+1;

data = zeros(n_elem,length(y11)/op+1);
if ang>0
   n=1;
   for ui = 1:1:n_elem
      ddata = y11(samp1(ui):end);
      ddata = resample(ddata,1,op);
      data(n,1:length(ddata)) = ddata;
      n=n+1;
   end
elseif ang<0
   m=n_elem;
   for ui = 1:1:n_elem
      ddata = y11(samp1(ui):end);
      ddata = resample(ddata,1,op);
      data(m,1:length(ddata)) = ddata;
      m=m-1;
   end
elseif ang==0
   for ui = 1:1:n_elem
      data(ui,1:length(y1)) = y1.';
   end
end

data = data(:,1:end-max(samp1));
```

```
if rem(length(data),2)~=0
   data = data(:,1:end-1);
end
```

A.4 bform_b5.m

```
function [conv1] =
bform_v5(X,angs,elem_d,c,fs,fc_old,fc_new,n_elem,cali_tau,cali_amps)

%X: data
%angs: [min angle, step angle, max angle]
%elem_d: distance between elements
%c: speed of sound
%fs: sampling frequency
%fc_new, fc_old: frequency desired, use same for both
%n_elem: number of elements on the array
%cali_tau, cali_amps: calibration constants
% nfft = 2.^(ceil(log(length(X))./log(2)));

nfft = length(X);
nifft = length(X);
bottom=(fs/2)/(nfft/2);
p_MID=fc_new/bottom; p_index=[p_MID*.95,p_MID*1.05];
ang_start=angs(1); ang_space=angs(2); ang_end=angs(3);
X2 = zeros(size(X));

for ui=1:n_elem
   X2(ui,:)=cali_amps(ui).*X(ui,:);
end

Xfft=fft(X2.',nfft); Xfft1=Xfft(1:nfft/2,:);
count=1; conv1=zeros(1,length(ang_start:ang_space:ang_end));

for jj=ang_start:ang_space:ang_end
   W=zeros(1,n_elem);
   %step through elements in steering vector calc

   for iji = 1:n_elem
      %calc the time delay and phase shift for the elements
      tau = ((iji-1)*elem_d./c).*sin(jj*pi/180);
      W(iji) = exp(-(j)*2*pi*(fc_old)*(tau+cali_tau(iji)));
%         W(iji) = exp((j)*2*pi*(fc_old)*(tau+cali_tau(iji)));
   end

   beam=ifft(W*Xfft1.',nifft);
   beam = real(beam)/(n_elem);
   [P1] = pwelch(beam,[],[],nfft,fs);
   conv1(count)=sum(P1(round(p_index(1)):round(p_index(2)),1));
   count=count+1;
end
```

A.5 bform_ang3.m

```
%% Beamform at Specific Angle
% Inputs:
%   X : matrix of collected data
%   ang : angle to be beamformed at
%   elem_space : spacing between elements
%   c : speed of sound
%   fc : center frequency to be beamformed at
%   n_elem : number of elements
%   cali_tau : phase calibration constants for fc
%   cali_amps : amplitude calibration constants for fc

function [beamed] = bform_ang3(X,ang,elem_space,c,fc,n_elem,cali_tau,cali_amps)

% nfft = 2.^(ceil(log(length(X))./log(2)));
nfft = length(X);
nifft = length(X);
X2 = zeros(size(X));

for ui=1:n_elem
   X2(ui,:)=cali_amps(ui).*X(ui,:);
end

Xfft=fft(X2.',nfft); Xfft=Xfft(1:nfft/2,:);
W=zeros(1,n_elem);

%step through elements in steering vector calc
for iji = 1:n_elem
   %calc the time delay and phase shift for the elements
   tau = ((iji-1)*elem_space./c).*sin(ang*pi/180);
   W(iji) = exp(-(j)*2*pi*(fc)*(tau+cali_tau(iji)));
end

% beamed=ifft(W*Xfft.',2*nifft);
% beamed = real(beamed(1:nifft))/(n_elem);
beamed=ifft(W*Xfft.',nifft);
beamed = real(beamed)/(n_elem);
beamed = beamed./max(beamed).*max(max(X));
```

A.6 peakfinder2D.m

```
%will find a set number of peaks in a plot

function [peaks] = peakfinder2D(dataplot, numPeaks, dist, thresh)
%dataplot: data
%numPeaks: number of peaks you wish to find
%dist: minimum distance between peaks
%thresh: minimum percentage of the max peak to look as decimal (ex: use 0.5
%   to find all peaks above 50% of max peak

   maxval = max(dataplot);
   n = 1;
   Xm = find(dataplot >= maxval*n);
   scoord(1) = Xm;
   m = 1;

   while (m <= numPeaks & n > thresh)
      n = n - 0.01;
      Xm = find(dataplot >= maxval*n);

      for k = 1:length(Xm)
         a = length(scoord);
         count = 0;

         for r = 1:a
            if ((abs(Xm(k) - scoord(r)) < dist));
               count = 1;
            end
         end

         if count == 0
            scoord(a + 1) = Xm(k);
            m = m + 1;
         end
      end
   end

   peaks = scoord;
```

A.7 wave.m

```
function [newout]=wave(sig, omw, smin, smax, ds, linr, shift, normal);

%WAVE       Wavelet Transform
%       WAVE(sig, omw, smin, smax, ds, linr, shift, normal) computes wavelet
%       transform with with the following parameters:
%
%               sig     input signal
%               omw     mother wavelet
%               smin    minimum scale
%               smax    maximum scale
%               ds      scale increment
%               linr    1 for linear scale increments, 0 for logarithmic
%               shift   1 omw centered at time zero, 0 first sample anchored at time zero
%               normal 1 for normalized, 0 for raw
%
% cfb 4 jun 1996

tic

global scaleindex
global delayindex
scaleindex=0;

if nargin == 1, omw=sig;, shift =1, linr=1, smin=.8, smax=1.2, ds=.01, normal=1, end
if nargin == 2, shift =1, linr=1, smin=.85, smax=1.15, ds=.01, normal=1, end
if nargin == 5, linr=1, shift=1, normal=1, end

lomw=length(omw);

omask = ones(1,lomw);
sigenergy=(abs(sig.^2));
```

```
omwenergy=sqrt(omw*omw');

lsig=length(sig);
numscales=round((smax-smin)/ds);
scalemult=(smax/smin)^(1/(numscales-1));

pwr=0;
pwrflag=1;
tlen=lomw/smin+lsig+1;
while pwrflag==1
pwr=pwr+1;
if 2^pwr >= tlen, pwrflag=0;,end;
end
fftlen=2^pwr;
pwrflag=1;

delayindex=[1:1:fftlen];
if linr==1,scaleindex=smin:ds:smax;,end

q=0;

if linr==1
 for scale = smin:ds:smax
  scale;
  q=q+1;
  %subsampling--time domain compression if scale > 1
  %time domain dilation if scale < 1
   mw=zeros(1,round(lomw/scale));
   lmw = length(mw);

   for k = 1:floor(lomw/scale)
    mw(k)=sqrt(scale)*omw(round(scale*k));
   end

   mask=ones(1,length(mw));

   newsigenergy=[sigenergy zeros(1,fftlen-length(sigenergy))];
   newsig=[sig zeros(1,fftlen-length(sig))];
   newmw =[mw zeros(1,fftlen-length(mw))];
   newmask = [mask zeros(1,fftlen-length(mask))];

   if shift==1
    newmw = [newmw((round((length(mw)/2))+1):length(newmw)),
newmw(1:round((length(mw)/2)))];
    newmask = [newmask((round((length(mw)/2))+1):length(newmask)),
newmask(1:round((length(mw)/2)))];
```

```
%        newmw=cbrot(newmw,round((length(mw)/2)));
%        newmask=cbrot(newmask,round((length(mw)/2)));
   end

   out=ifft(fft(newsig,fftlen).*conj(fft(newmw,fftlen)));
   norm=ifft(fft(newsigenergy,fftlen).*conj(fft(newmask,fftlen)));
   clear newsig newmw

%  if ftshift==1
%  wavtran(q,:)=fftshift(out(1:lsig));
%  normtran(q,:)=fftshift(norm(1:lsig));
%  else
   wavtran(q,:)=out(1:lsig);
   normtran(q,:)=norm(1:lsig);
% end
 end

else
 for q=1:numscales
  scale = smin*(scalemult)^(q-1);
  scaleindex(q)=scale;
  %subsampling--time domain compression if scale > 1
  %time domain dilation if scale < 1
   mw=zeros(1,round(lomw/scale));
   lmw = length(mw);

   for k = 1:floor(lomw/scale)
     mw(k)=sqrt(scale)*omw(round(scale*k));
   end

%
%   newsig=[sig zeros(1,fftlen-length(sig))];
%   newmw =[mw zeros(1,fftlen-length(mw))];
%

   mask=ones(1,length(mw));

   newsigenergy=[sigenergy zeros(1,fftlen-length(sigenergy))];
   newsig=[sig zeros(1,fftlen-length(sig))];
   newmw =[mw zeros(1,fftlen-length(mw))];
   newmask = [mask zeros(1,fftlen-length(mask))];

%   if (rem(length(sig),2))==0
%        newmw = cbrot(newmw,length(mw)/2);
%       newmw = [newmw((length(mw)/2+1):length(newmw)),
newmw(1:length(mw)/2)];

%   else
%        newmw = cbrot(newmw,fix(length(mw)/2));

  if shift==1
    newmw = [newmw((round((length(mw)/2))+1):length(newmw)),
newmw(1:round((length(mw)/2)))];
    newmask = [newmask((round((length(mw)/2))+1):length(newmask)),
newmask(1:round((length(mw)/2)))];
  end
%   end

%  out=ifft(fft(newsig,fftlen).*conj(fft(newmw,fftlen)));
   out=ifft(fft(newsig,fftlen).*conj(fft(newmw,fftlen)));
   norm=ifft(fft(newsigenergy,fftlen).*conj(fft(newmask,fftlen)));

%  if ftshift==1
%   wavtran(q,:)=fftshift(out(1:lsig));
%  else

%   wavtran(q,:)=out(1:lsig);
   wavtran(q,:)=out(1:lsig);
   normtran(q,:)=norm(1:lsig);
%  end
 end
end

 if normal == 1
  newout=abs(wavtran)./(sqrt(abs(normtran))*omwenergy);
 else
  newout=wavtran;
 end

a=toc;
time=a/60;
```

A.8 spikemaker.m

```
function [newoutT] = spikemaker(data, numPeaks, dist, thresh, G, c, fs)
```

```
%data: data
%numPeaks: number of peaks you wish to find
%dist: minimum Y distance between peaks
%thresh: minimum percentage of the max peak to look as decimal (ex: use 0.5
%   to find all peaks above 50% of max peak
%
% xg = 0:.1:10; yg = 0:.1:10; xgo = 5; ygo = 5;
% for c1 = 1:length(xg)
%     for c2 = 1:length(yg)
%         G(c1,c2) = exp(-((((xg(c1)-xgo).^2)/(2*5))+(((yg(c2)-ygo).^2)/(2*5))));
%     end
% end

maxval = max(max(data));
D = size(data);
ttt = size(G);
spike = zeros(D(1),D(2));
%spike = spike./5;
targetsize = dist/c*fs;
n = 1;
[Xm, Ym] = find(data >= maxval*n);
scoord(1,:) = [Xm, Ym];
m = 1;

%note that this will not check the number of peaks or the percentage
%   threshold until after it has ran through the current list of peaks,
%   meaning that the resultant peak list could be larger than m.

while (m <= numPeaks & n > thresh)
   n = n - 0.01;
   [Xm, Ym] = find(data >= maxval*n);
   for k = 1:length(Xm)
      [a,b] = size(scoord);
      count = 0;
      for r = 1:a
         if ((abs(Ym(k) - scoord(r,2)) < targetsize));
            count = 1;
         end
      end
      if count == 0
         scoord(a + 1,:) = [Xm(k), Ym(k)];
         m = m + 1;
      end
   end
end
```

```
[a,b] = size(scoord);
for g = 1:a
%>>>>>>>>>>>>>>>>WATCH ACCURACY HERE IF YOU CHANGE SIZE OF G
   if scoord(g, 2) - ((ttt(1)-1)/2) <=0
      spike(scoord(g, 1),scoord(g, 2)) = data(scoord(g, 1),scoord(g, 2));
   else
      spike(scoord(g, 1),scoord(g, 2) - ((ttt(1)-1)/2)) = data(scoord(g, 1),scoord(g, 2));
   end
   %the (ttt(1)-1)/2 part of the equation ensures the peak of the gaussian curve is in the
correct location instead of the curve starting in the correct location
end

newoutT = conv2(G, spike);
%newoutT = spike;
```

A.9 peakfinder3D.m

```
%will find a set number of peaks in a plot

function [peaks] = peakfinder3D(dataplot, numPeaks, dist, thresh)

%dataplot: data
%numPeaks: number of peaks you wish to find
%dist: minimum Y distance between peaks
%thresh: minimum percentage of the max peak to look as decimal (ex: use 0.5
%   to find all peaks above 50% of max peak

maxval = max(max(dataplot));
n = 1;
[Xm, Ym] = find(dataplot >= maxval*n);
scoord(1,:) = [Xm, Ym];
m = 1;

%note that this will not check the number of peaks or the percentage
%   threshold until after it has ran through the current list of peaks,
%   meaning that the resultant peak list could be larger than m.

while (m <= numPeaks & n > thresh)
   n = n - 0.01;
   [Xm, Ym] = find(dataplot >= maxval*n);
```

```
    for k = 1:length(Xm)
        [a,b] = size(scoord);
        count = 0;

        for r = 1:a
            if ((abs(Ym(k) - scoord(r,2)) < dist));
                count = 1;
            end
        end

        if count == 0
            scoord(a + 1,:) = [Xm(k), Ym(k)];
            m = m + 1;
        end
    end
end

peaks = scoord;
```

A.10 maxvalAngles.m

```
function [vals] = maxvalAngles(worldData)

for i = 1:91
    vals(i) = max(max(worldData(:,:,i)));
end

%vals(15) = 10000;

figure, plot(vals)
```

A.11 combiner.m

```
for i = 1:24
    if i == 1
    load WORLDTESTFILE1.mat;
    load WORLDTESTFILE2.mat;
    bigworld = WORLD1.*WORLD2;
    maxvals1 = maxvalAngles(WORLD1);
    F(1) = getframe;
    maxvals2 = maxvalAngles(bigworld);
    F(2) = getframe;
    clear WORLD1 WORLD2
else
    openString = sprintf('load WORLDTESTFILE%d.mat',i);
    eval(openString);
    multString = sprintf('bigworld = bigworld.*WORLD%d;',i);
    eval(multString);
    maxvalString = sprintf('maxvals%d = maxvalAngles(bigworld);',i);
    eval(maxvalString);
    F(i) = getframe;
    clearString = sprintf('clear WORLD%d',i);
    eval(clearString);
end
        end
```