Optimization of Transmit/Receive Array Topology in 3D Acoustic Imaging

by

Robert M. McPhie

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

February 4, 2003

Author.....
Department of Electrical Engineering and Computer Science
February 4, 2003

Certified by.....
Kenneth Erikson
VI-A Company Thesis Supervisor

Certified by.....
Markus Zahn
M.I.T. Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Optimization of Transmit/Receive Array Topology
in 3D Acoustic Imaging

by
Robert M. McPhie

Submitted to the Department of Electrical Engineering and Computer Science
February 4, 2003
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Acoustic imaging transmit and receive arrays pairs are simulated in the bistatic transmission case using Field II, an industry standard ultrasound toolbox. Analysis of imaging fundamentals, as well as simple transmit and receive array beam patterns reveals desirable beam pattern properties that can be achieved with very dense arrays. As high element numbers in acoustic arrays are cumbersome with respect to manufacture and signal processing, techniques for reduction of array element numbers are reviewed. Array optimization is shown to be both highly desirable and attainable through straightforward iterative simulated annealing methods. A highly flexible array cost measure allows optimized array pairs to be generated based on a user-defined set of desired array properties.

M.I.T. Thesis Supervisor: Markus Zahn
Title: Professor of Electrical Engineering

VI-A Company Thesis Supervisor: Kenneth Erikson
Title: Senior Principal Systems Engineer, BAE SYSTEMS

Work sponsored by Office of Naval Research, ONR 321SS.

Acknowledgments

First I would like to thank Ken Erikson for his relentless support of my Thesis work. He took the initiative to find existing needs at BAE SYSTEMS and adapt them to be excellent research endeavors. Not only was he the key player in helping to find a thesis topic, but he consistently provided me with excellent support, both intellectually and logistically. Thanks also go to Geoff Edelson, Rob Nation, Mike Shinego, and Jason Stockwell for their support of me and the VI-A Internship Program at BAE SYSTEMS.

Without the aid of Professor Markus Zahn, I would have had no opportunity to do this thesis work at all. He went the extra mile to give me a chance I couldn't have gotten alone, and I am very thankful for his advocacy on my behalf.

Furthermore, I can't thank my parents enough for their constant encouragement and support of me at MIT from beginning to end. Their belief that I could succeed was a powerful motivation for me during difficult times.

Finally, I would like to express my gratitude for my sweet wife Megan, who sacrificed her time, effort, and convenience right along with me during the writing of this thesis. As she put up with my many long nights, the extra computers in the apartment, and all the attention they received that should have been hers, she put my needs above her own. For her loving support, I am deeply grateful.

Contents

List of Figures

List of Tables

1 Introduction to Acoustic Imaging

Advances in imaging technology have been primarily concerned with making the unseen world visible. Acoustic imaging has its roots in the development of sonar devices for the detection of submarines during World War I. This technology, pioneered in France, proved useful in keeping German U-boats in check [1]. Further development of the techniques used for acoustic imaging have yielded widely varying applications.

1.1 Acoustic vs. Photographic Imaging

1.1.1 Material Transparency

The strength of acoustic imaging over more developed and conventional photographic imaging methods lies in the opacity of everyday materials. An object of interest might be completely occluded from vision by a material that is not transparent to light, but is transparent to sound. For example, an aquatic mine in murky water might be completely hidden from view with even very powerful lights, but easily detected with ultrasound. X-rays may be risky to use in the monitoring of a fetus in the mother's womb, but acoustic pulses are completely safe.

1.1.2 Lenses

Passive acoustic lenses have been in use for decades, but the attenuation of acoustic waves through a solid lens is often great, and therefore unacceptable. Recent uses of fluid-filled lenses with acoustic matching films have promised to allow the use of acoustic lenses in longer-range and lower-power applications.

1.1.3 Wavelength

Characteristically, photographic and acoustic imaging have a fundamental difference in the wavelengths of their respective sources of "illumination." This presents some interesting differences in the capabilities of the different imaging techniques. The difference arises due to the dramatic difference in wave speeds in most useful media as shown in Table 1. This causes a corresponding dramatic difference in the wavelengths of electromagnetic and acoustic waves of comparable frequencies. Both

	Wave Speed in Medium (m/s)	
Medium	EM Wave	Acoustic Wave
Vacuum	299,792,458	—
Air	299,702,547	330
Water	225,407,863	1,480
Avg. Human Soft Tissue	—	1,540

Table 1: Wave speeds in various mediums

imaging methods can use a wide spectrum of frequencies, giving rise to a wide range of wavelengths available as shown in Tables 2 and 3. Here, there is overlap in the

		Wavelength (μm)	
Type of EM Wave	Frequency (Hz)	Air	Water
Infrared	3×10^{12} - 3×10^{14}	1-100	0.75-75
Visible	4.3×10^{14} - 7.5×10^{14}	0.4-0.7	0.3-0.53
Ultraviolet	7.5×10^{14} - 7.5×10^{16}	0.004-0.4	0.003-0.3
X-Ray	1×10^{17} - 1×10^{19}	0.00003-0.003	0.00002-0.002

Table 2: Electromagnetic waves commonly used in imaging

	Wavelength (μm)		
Frequency (Hz)	Air	Water	Tissue
3×10^{4}	11,000	49,000	51,000
3×10^{5}	1,100	4,900	5,100
3×10^{6}	110	490	510
3×10^{7}	11	49	51

Table 3: Acoustic Waves Commonly Used in Imaging

wavelengths of infrared light and ultrasound in the 10MHz range. So it is that recent developments in IR focal plane development technologies have provided interconnect systems useful for multidimensional acoustical arrays as well [2].

1.2 Considerations Affecting Array Topology

1.2.1 Bi-static vs. mono-static

Bi-static mode is when an energy source separate from the focal plane ensonifies the target, the waves bounce off targets in the area, and the echoes are received at the array. In mono-static mode, the array itself transmits the pulse and receives the echo. Inherently it is advantageous to use mono-static transmission in terms of image resolution. By directing the pulse to a specific area to be imaged, there is less "background noise" than when a single source ensonifies the entire scene. The drawback is the complexity of not only coordinating the received waves from many small transducers, but transmitting all of them in a coordinated fashion and with a decent amount of power as well. High voltage switching greatly increases the amount of support electronics needed to run an acoustic array.

A compromise between the two techniques, available primarily for use with larger sonar arrays, is to have a separate array of transmitters and receivers. These can be interleaved in a myriad of different configurations over a flat or curved surface. Switching is not required, but the transmit and receive beams can still both be accurately steered. The transmit and receive electronics can be separated and optimized independently for their specific requirements. This thesis will address the problem of optimizing the physical layout of a transmit array and receive array to work in concert.

1.2.2 Sidelobes and Grating Lobes

Sidelobes exist in the beam characteristics of an array based on the frequency used. These are also a result of the geometry of the array and the effective apodization of the aperture. By choosing an aperture with a magnitude characteristic that changes smoothly over its surface, these side lobes can be reduced, at the expense of a reduction in beamwidth resulting in resolution loss. Grating lobes in the array characteristic are also present due to the regular spatial interval of small transducers, and can be reduced by using a more sparse, quasi-random array.

1.2.3 Frame rate

Resolution in the depth dimension has much to do with frame rate. Because acoustic waves travel slowly (with respect to electromagnetic waves) there are only so many pulses that can transmitted, echoed, received, and processed in a typical real time frame of 1/30 of a second. Some systems have been designed using two or more pulse-echo systems multiplexed to allow multiplication of frame rate.

2 Simulation Methods

2.0.4 Huygen's Principle

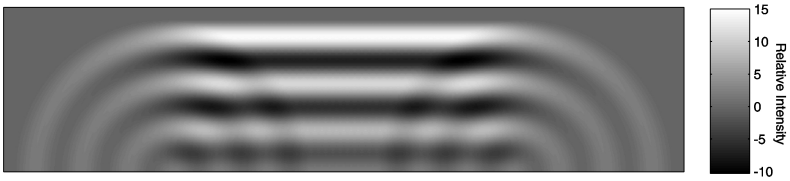
In 1678, the Dutch physicist Christian Huygens published a theory regarding wave propagation. One of the principles proposed by Huygens was that any wavefront can be described as a surface of radiating point sources, and furthermore that the propagation of that wavefront can be determined by summing the spherical radiation patterns of all of the point sources on the wavefront surface. Consider the simulated radiation pattern of a single radiating point source as in figure 1. To model the propagation of a plane wave with the same excitation, this single point is superimposed upon itself after a spatial displacement. The patterns for 10 and 50 iterations are also shown in Figure 1 using different spacing. The model using 50 points is an excellent approximation to the true propagation of such a wave from a planar source. In general, as a finer spatial sampling is used, a better approximation is achieved. Huygen's Principle (as it is now known) is of great use in modeling the wave patterns generated by arrays of sources. This theory provides the basis for the Field II simulation package used to do array optimization.

2.1 Field II Background

The Field II Simulation Program was developed by Joergen Arendt Jensen of the Technical University of Denmark. The package consists of modules used in a Matlab environment for array simulation. Using the concept of the spatial impulse response together with linear systems theory allows the simulator to calculate the emitted acoustic field for a transducer array of arbitrary geometry. By subdividing individual

(a)

(b)



(c)

Figure 1: Simulated radiation patterns for (a) a single point source, (b) 10 points at 0.85λ spacing, and (c) 50 points at 0.15λ spacing

array elements, far-field approximations are maintained. This, in turn, allows for simplified calculations and reduced computation time [3].

2.1.1 Field II Simulation Example

A 25 element, linearly-spaced, 1D array is shown in Figure 2. The spacing is roughly 2λ using a 40kHz excitation pulse. Each element is 25.4mm in diameter. This array was used as a benchmark for initial system testing and performance as the problem of where to put 25 elements in a 50λ wide aperture is a classic problem that has gotten attention for decades [4]. Each physical element in the array was further subdivided into 6 mathematical elements to strengthen the far-field approximation. Field II calculates the appropriate delay times for all elements with a specified array focal point 100m directly normal to the array. For purposes of simplification, each element is assumed to have an impulse response of the unit impulse.

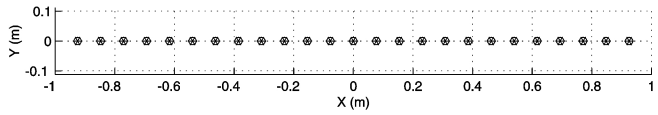


Figure 2: 25 element 1D array

Desiring to model this array in the CW case, an excitation pulse of 25 cycles was used with a frequency of 40kHz as shown in Figure 3. Finally, the system sampling frequency was set to 64 times the pulse frequency and the wave speed was set to 1480m/s. This is a typical speed for sound in water.

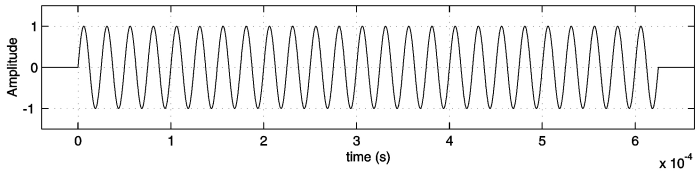


Figure 3: Transmit array excitation pulse

Figure 4 shows the raw output data from Field II measured at 100m. Each column of output data represents the received signal at a distinct point in space in front of

the array in a specific time interval. For virtually all simulations, measurements were made at points along an arc at a fixed distance from the center of the array. Of note in Figure 4 is the center dotted line corresponding to the received signal at the focal point of the array. This mirrors the array excitation pulse shown in Figure 3 showing that the array was indeed properly focused at the desired point. Also of note are the vertical bands at $\pm 30^\circ$ from the center as well as the faint bands at $\pm 75^\circ$. These bands indicate that an array with 2λ spacing will emit significant amounts of energy at angles other than the main steer angle.

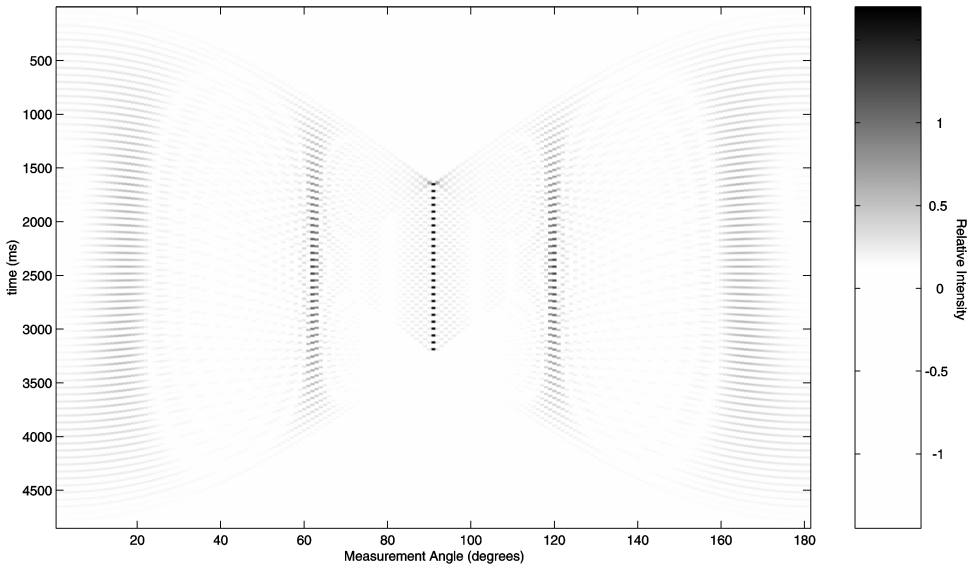


Figure 4: Field II raw output data

The usefulness of the Field II simulation engine becomes clearer when this raw data is further analyzed, and that analysis is used to determine how to make desirable changes to the array topology. Preliminarily, however, it is necessary to convert this raw data into a relatively few parameters which are indicative of array performance.

2.2 Array Plotter Interface

A comprehensive interface was needed for visualization of the input arrays being simulated by Field II as well as to analyze the output. This was built using Matlab

user interface tools. Of primary interest was the ability to see an acoustic array topology of interest, set the parameters of the simulation, and then visualize the simulated beam pattern in an intuitive manner.

2.2.1 Array Configuration

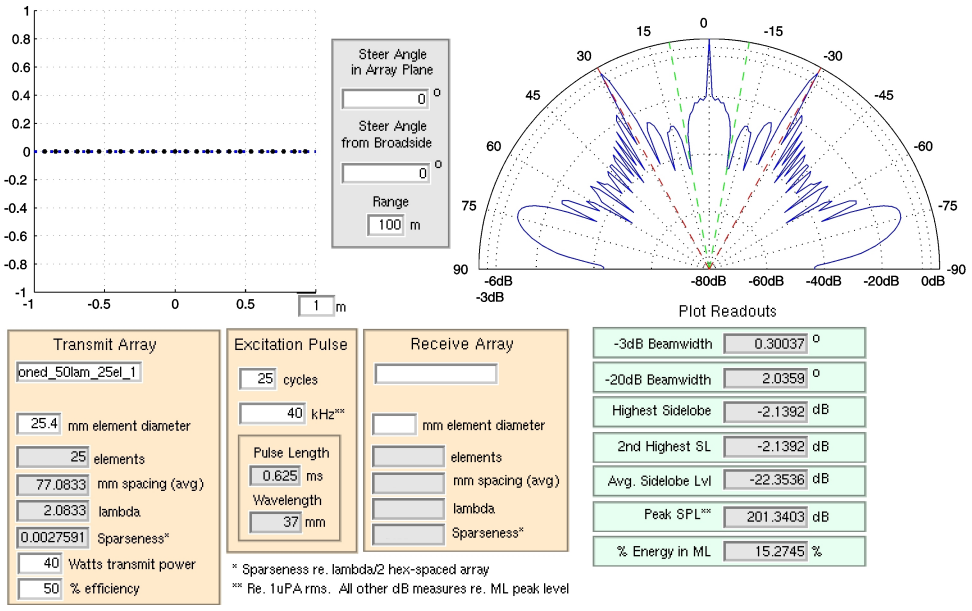


Figure 5: Screenshot of Array Plotter user interface showing array configuration (upper left), transmit and receive array parameters (lower left), angles used for beam steering and beam plot (upper middle), plot of beam pattern (upper right), and resultant beam parameters (lower right)

Figure 5 shows the interface which was developed to allow a user to do accomplish those objectives. Seen in the upper-left of the screen is the physical array layout plot. Here the 25 element array from Figure 2 is shown again. Beneath the plot are parameters which are user-defined (summarized in table 4), as well as data for user information. Listed are 2 different measures of array spacing. The first, average spacing, is computed individually for all elements by averaging the distance to the 2 closest elements. This measure for each element is then averaged over the entire array. Effectively, this indicates how much clustering there is in the array, since a

sparse array will still yield a small value if elements are in clusters of 3 or more. The second array spacing measure is a sparseness measure, which computes the ratio between the actual number of array elements, and the number of elements in a circular aperture of the same diameter densely populated with elements in a hexagonal pattern at $\lambda/2$ spacing. This measure is most useful when the array being simulated is a 2D array with a roughly circular aperture. Both measures use the current pulse frequency value to give data relative to the pulse wavelength.

The user-specified angles shown in the upper-middle portion of Figure 5 determine the focal point for the transmit and receive arrays. The first angle is the desired beam steer angle measured from the horizontal axis in the array plane shown in the upper-left plot. This angle is shown on the plot for user reference as a blue dotted line (horizontal in Figure 5). The beam pattern for the array is calculated in the plane that includes this line as well as the second user-determined angle. This next angle specified is the desired beam steer angle measured from a vector normal to the surface of the array (broadside). This angle of steering should subsequently be visible in the main lobe of the beam pattern plot in the upper-right portion of Figure 5.

2.2.2 Beam Pattern Plotting and Analysis

The upper-right portion of the Array Plotter interface in Figure 5 is a polar 2D reduction plot of the 3D data returned by Field II. This is generated by detecting the temporal center of the received pulse at the array steer angle. The working data is restricted to a window of time of 1 wavelength. An RMS calculation is done on this window for each angle to obtain a single vector containing the transmit level at that angle. These values are plotted on a polar grid using a customized plotting routine to handle polar logarithmic plotting. The built-in Matlab plotting routines are capable of handling positive values only, and there is little provision for logarithmic polar plotting. Thus, a graphics function for Matlab was developed to take data from Field II (or any polar data), use the peak value as the 0dB level, and plot the data logarithmically from -90° to +90°.

The beam pattern is then analyzed to provide the readouts below the beam plot. First, the -3dB and -20dB values are calculated. The resolution of these values was initially programmed to be restricted to increments of the beam plot resolution. To allow

User Input	Description
Filename	ASCII file containing element center point coordinates
Element Diameter	Element diameter for all elements in array
Transmit Power	Determines the transmit array settings in Field II
Transmitter Efficiency	Determines the transmit array settings in Field II
Pulse Frequency	Pulse frequency used for transmitter excitation
Pulse Cycles	Number of cycles of transmit pulse
Steer Angle - Array Plane	Angle from horizontal (in array plane)
Steer Angle - Broadside	Angle from array normal to array focal point
Range	Distance from array center point to array focal point

Table 4: User inputs to Array Plotter interface

for finer perception of improvement or degradation of these values, however, a linear interpolation scheme was later applied. This takes advantage of the high-resolution floating point values computed in the RMS calculations, and is of importance in optimization. It should be noted that this linear interpolation suffers from inaccuracy when actual beamwidths are much less than the plot resolution. As a 1° plot resolution was used for all plots in this thesis, beamwidth readouts much less than 1° often represent slightly higher true beamwidths. However, any changes made to the true beamwidth will be reflected in the linearly-interpolated value, so this measure was deemed acceptable for use with optimization.

Next a peak-detecting function finds all maxima and minima in the beam plot. A threshold value of 3dB was used in peak-detection to remove many false maxima being detected by small oscillations in the beam pattern. The peak-detection function stores the values for the highest peaks and their angles, as well as the angle of the first minima on either side of the main lobe. Both sets of angles are shown graphically on the beam plot itself, the first in red indicating the highest 2 sidelobes, the second in green indicating the auto-detected angular range of the main beam. These minima angles are then used to determine the average sidelobe level (the average received response not including the main lobe) and the energy ratio measure (the integrated main lobe response divided by the integrated total response level).

Figure 5 clearly shows the high sidelobes at $\pm 30^\circ$. The plot readout identifies these as being 2.1dB below the main beam level. The plot shows the significant array response at $\pm 75^\circ$ as well. Both sets of sidelobes are undesirable array characteristics that optimization will attempt to minimize.

2.2.3 Receiving Array Case

Identical calculations to the transmit case are performed when analyzing the pattern received by an array of receiver transducers. Rather than measuring the received energy at an arc of points above a transmitting array, however, the plotted pattern is that of what is received when a point source is sequentially located at an arc of points above the receiving array. Due to the reciprocal nature of acoustic transducers, calculated beam patterns are identical whether considering them as resultant from transmitting arrays or receiving arrays. The differences seen in manufactured array

configurations arise chiefly due to differences in element size, manufacturing costs, and power requirements for transmitters compared to receivers.

2.2.4 Pulse-Echo Case

Some additional calculations are performed when calculating the pulse-echo response of a transmit/receive array pair. First, each beam pattern is calculated independently. Some data is stored from these separate beam patterns such as the angular location of the highest sidelobes of each and the transmit power received from the transmitting array. The two responses are then multiplied to yield a composite beam pattern showing the response of the system to a transmit pulse from the transmit array echoing off of the focal point of both arrays and being received by the receiving array. This pulse-echo scenario is representative of the majority of sonars and ultrasound imaging systems in wide use.

3 Simulation Results

3.1 1D Linear Array

The 1D linear array shown in the Section 3.1 illustrates a few pros and cons for that particular array topology. It can be seen that the beamwidths at -3dB and -20dB are 0.3° and 2.0° respectively, which are excellent for an underwater sonar system, for example. This resolution is attainable in the dimension parallel to the array itself. In the perpendicular direction, however, this array has no resolving power whatsoever, as shown in Figure 6. Hence, the usefulness for such an array is limited.

3.1.1 Modification to 1D Array

A fixed acoustic lens can be placed over a linear array. This helps to combat the problem of poor resolution in the elevation direction and is widely used in medical ultrasound. The drawback is that it adds bulk to the outside of a sensor, decreased sensitivity due to attenuation within the acoustic lens, and a fixed focus has no flexibility in dynamically choosing a desired depth of focus.

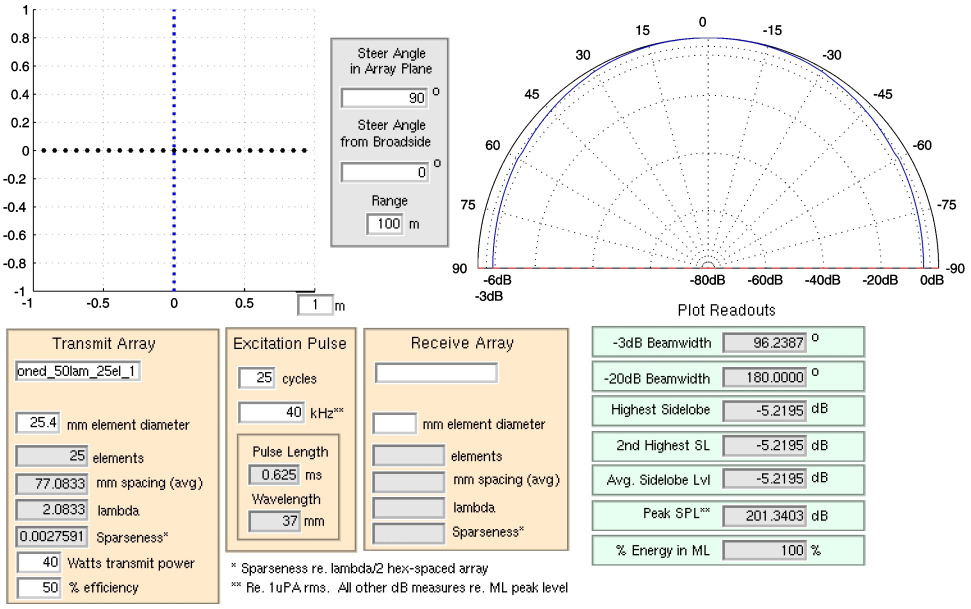


Figure 6: Lack of off-axis focusing ability in 1D array shown by wide beamwidth in beam pattern plot

3.2 1.5D Rectangular Aperture array imaging

The simple answer to the lack of off-axis resolution in the 1D array case is to use a 25x25 element 2D array rather than a 1x25 element array. Before considering that, however, attention should be paid to the intermediate solution. For some medical applications, for example, use of a 1D array results in blurred imaging, but a fully populated 2D array is either too cumbersome to handle or requires too much signal processing than is practical. A 2-dimensional array that extends only partially in the 2nd direction is known as a 1.5D array. Such an array offers the advantage of some focusing in the azimuthal direction, while not requiring the n^2 independent channels of a 2D array.

3.2.1 1.5D Medical Ultrasound Model

Before the arrayplotter interface was developed, some beam cross-section simulations were performed to assess the potential benefits of a 1.5D array for use in medical ultrasound. The goal was to show that combining array focusing with a fixed-focus

lens over a 1.5D array would allow a user to achieve greater resolution in the elevation direction, but also maintain flexible focusing capability. This desired effect is illustrated in Figure 7.

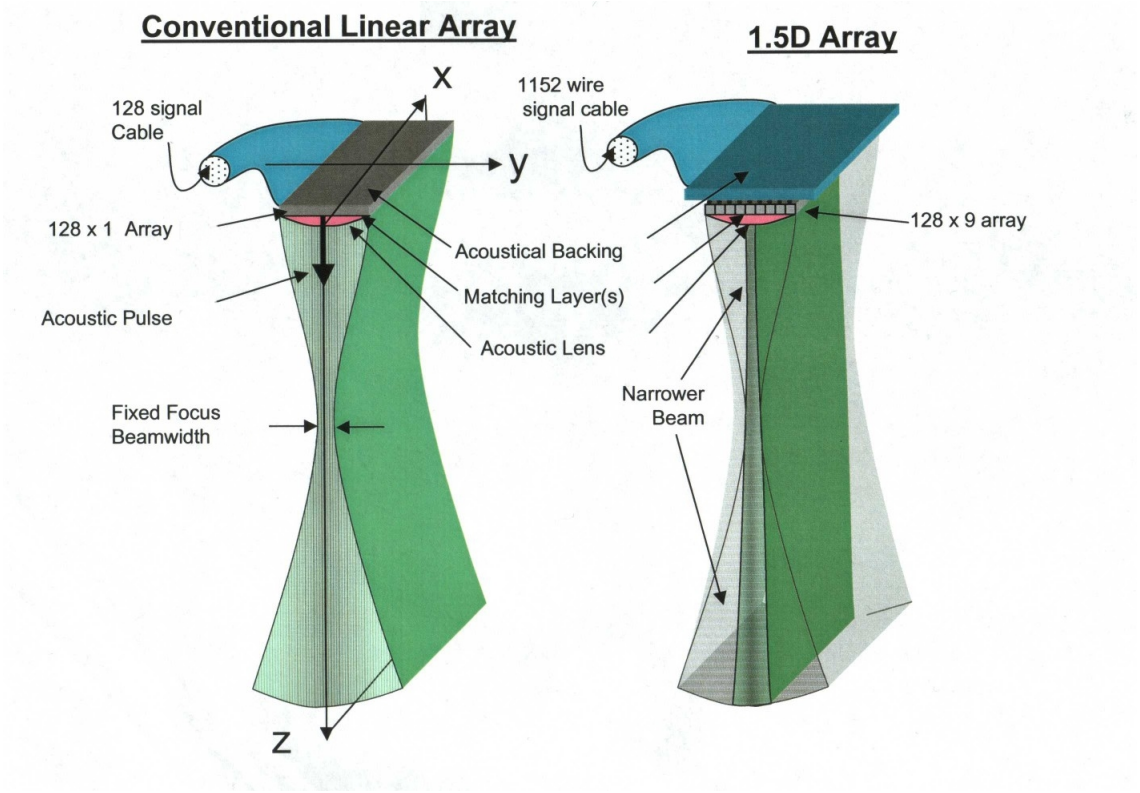


Figure 7: Proposed dynamic focusing capability of 1.5D array compared to fixed focus of conventional 1D linear array (Courtesy of K. Erikson et al. [9])

Shown in the figures that follow are beam cross-section simulations of a 128x9 pixel 1.5D array measuring 2.04cm wide and 1.35cm high. An acoustic lens was simulated which focused the beam in the elevation direction 5cm from the surface of the array. Simulations were performed with Field II using the array in mono-static transmission mode with a 5MHz center frequency. Shown in Figure 8, the lens focuses well at a depth of 5cm, but not as well at 3cm or 7cm. Figures 9 and 10 show that the electronic focusing can dramatically increase the flexibility of elevation resolution of a 1.5D array. Approximate beamwidths are shown in Table 5 showing that elevation beamwidths were substantially reduced at all depths using electronic focusing. Thus,

the variable focusing capability proposed for a 1.5D array (shown in Figure 7) is achievable.

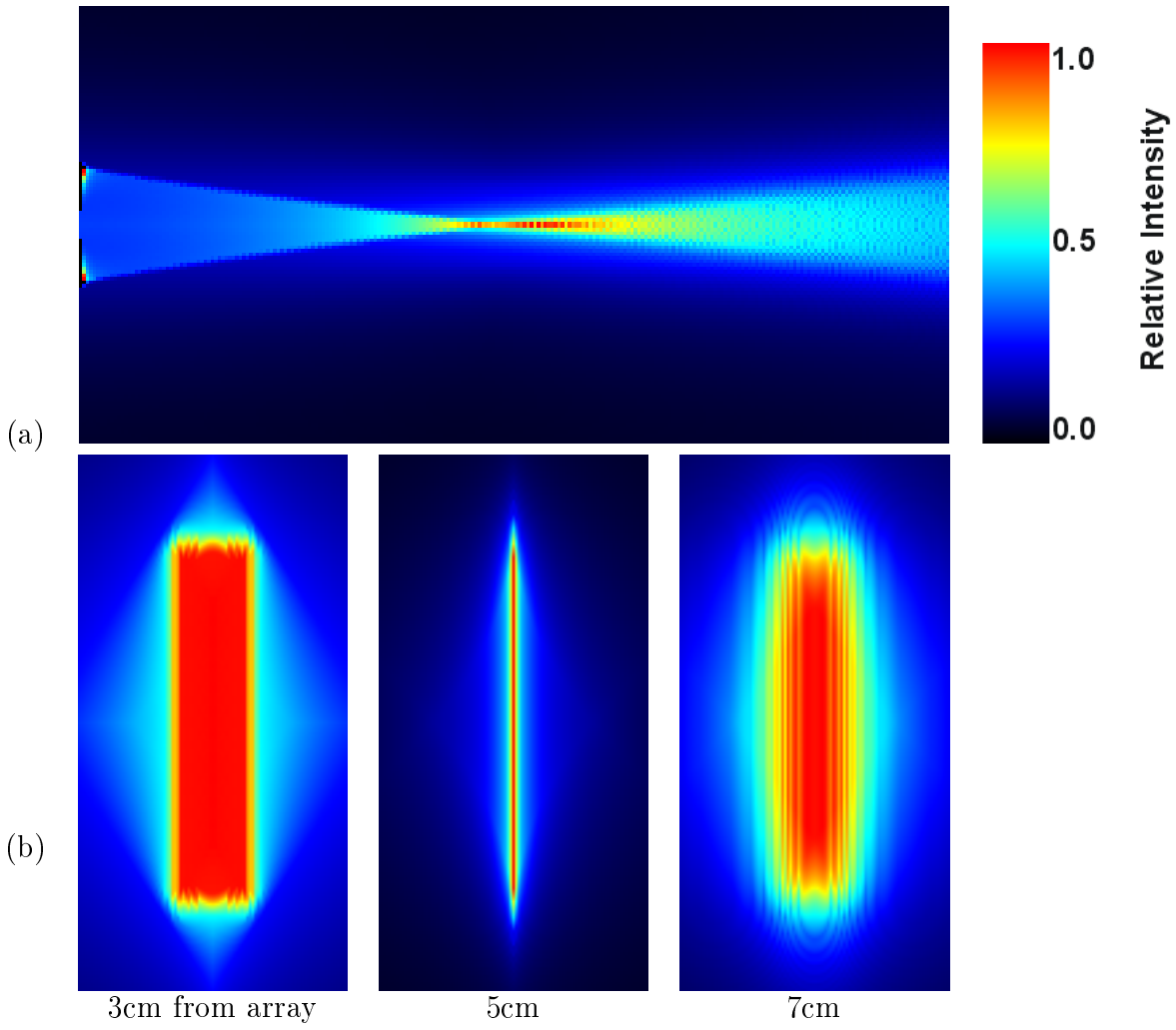


Figure 8: (a) Simulated beam profile normal to a 1.5D array with no electronic focusing (simulated array at plot left). (b) Corresponding simulated beam profiles parallel to the 1.5D array plane at specified distances from the array face. A fixed-focus acoustic lens with a 5cm focal length is present.

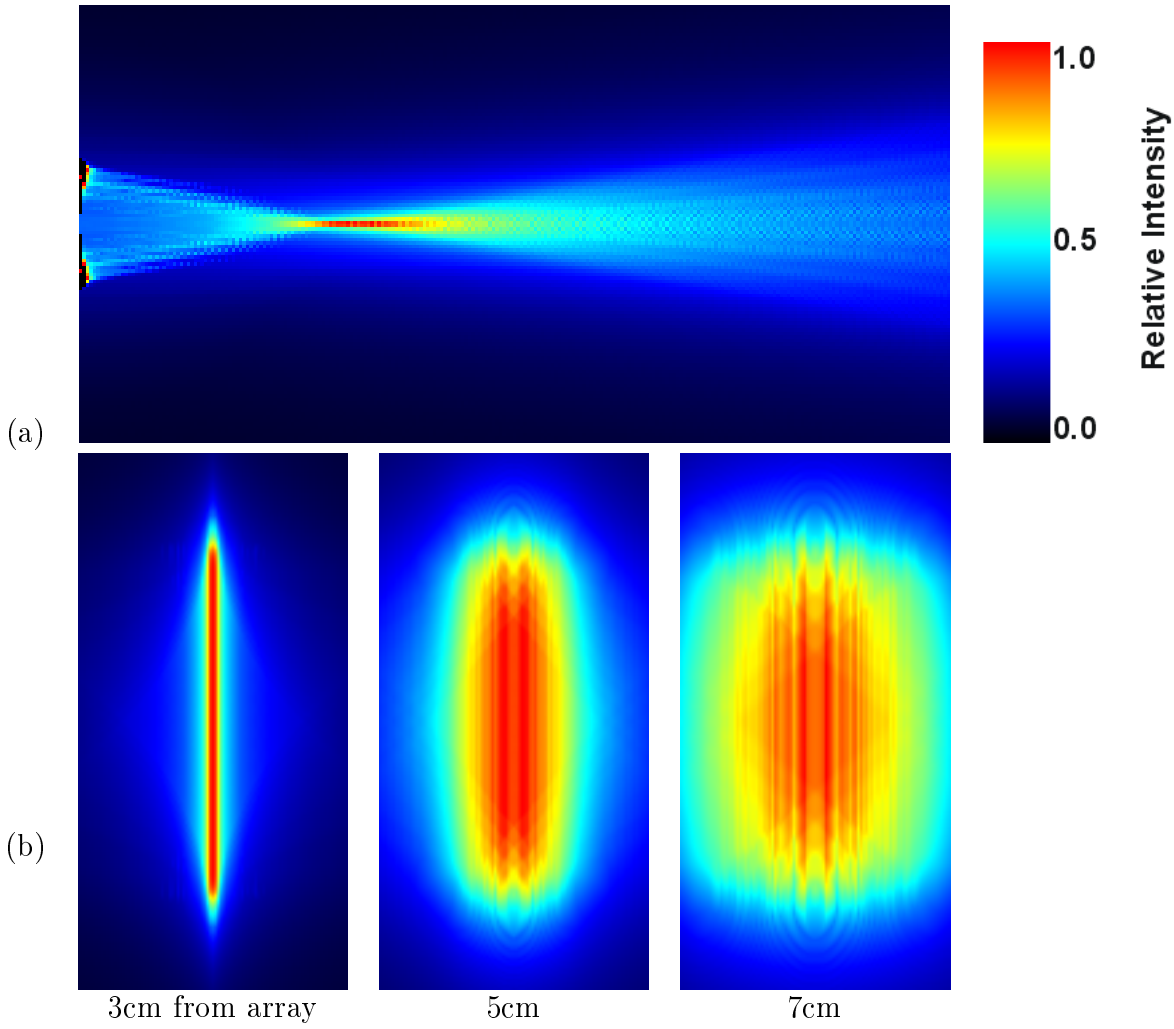


Figure 9: (a) Simulated beam profile normal to a 1.5D array electronically focused at 3cm from the array face. (b) Corresponding simulated beam profiles parallel to the 1.5D array plane at specified distances from the array face. A fixed-focus acoustic lens with a 5cm focal length is also present.

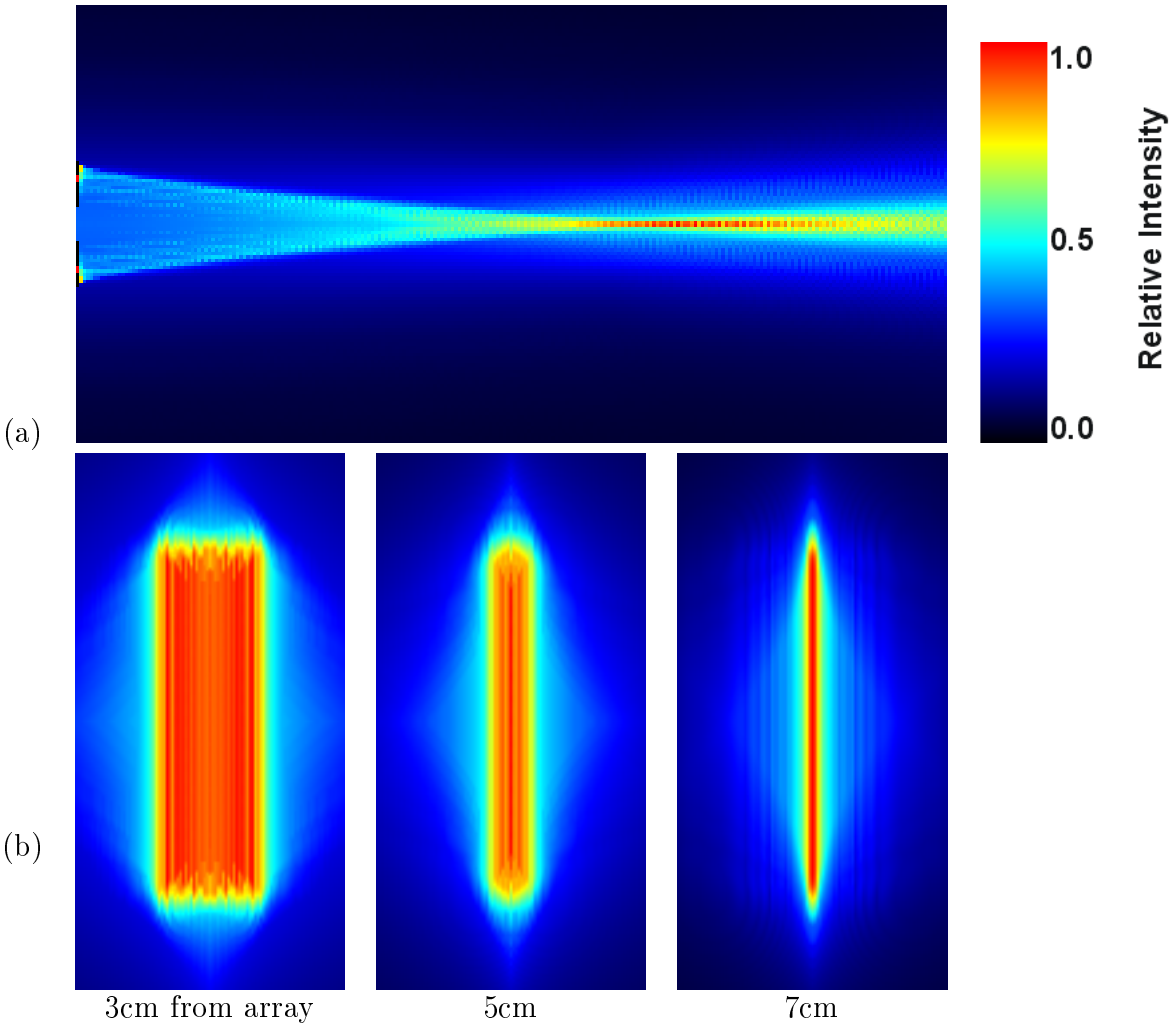


Figure 10: (a) Simulated beam profile normal to a 1.5D array electronically focused at 7cm from the array face. (b) Corresponding simulated beam profiles parallel to the 1.5D array plane at specified distances from the array face. A fixed-focus acoustic lens with a 5cm focal length was also present in the simulations.

	Approx. Beamwidth at Dist. From Array (mm)						
Array Focus	2 cm	3 cm	4 cm	5 cm	6 cm	7 cm	8 cm
2cm	0.75	2.25	3.15	5.40	7.20	7.50	8.70
3cm	2.85	0.60	1.65	3.75	4.35	6.30	8.10
4cm	4.95	1.95	0.45	2.10	4.05	6.75	7.65
5cm	6.45	4.05	1.65	0.30	1.80	3.15	5.85
6cm	7.20	4.65	2.85	1.05	0.60	1.05	2.55
7cm	7.50	4.95	3.30	1.65	0.75	0.60	1.35
8cm	7.95	5.55	3.75	2.55	1.65	0.75	0.75
min	0.75	0.60	0.45	0.30	0.60	0.60	0.75

Table 5: Simulated 1.5D array approximate beamwidths at specified depths showing array focusing capability

3.2.2 1.5D Array Simulation Results

Returning to the original 1D example of Section 3.1, by similarly adding 4 rows of 25 elements above the original 1D array, as well as 4 rows below, the off-axis resolution of the array is immensely improved. Figure 11 shows that the 96° beamwidth in Figure 6 has now been reduced to 2.5°.

3.2.3 Simplification of 1.5D Array Signal Processing

While the 1.5D array maintains simplicity with few array elements in the second dimension, further simplifications can be made in the method of processing the transducer signals. As shown in Figure 12, the transmit and receive signals to the column of 9 vertical elements is symmetrical about the middle element. This eases computation, as only 5 phase values for any particular depth must be calculated, instead of 9. Four signals are split and fed to transducers opposite each other. This also maintains accurate directional focusing in the elevation direction to be exactly in line with the normal direction of the middle transducer.

3.3 2D Rectangular Aperture Array Imaging

The gold standard of capable arrays is the fully-populated 2D array. Resolution in the elevation and azimuthal directions is equivalent. However, as previously discovered in

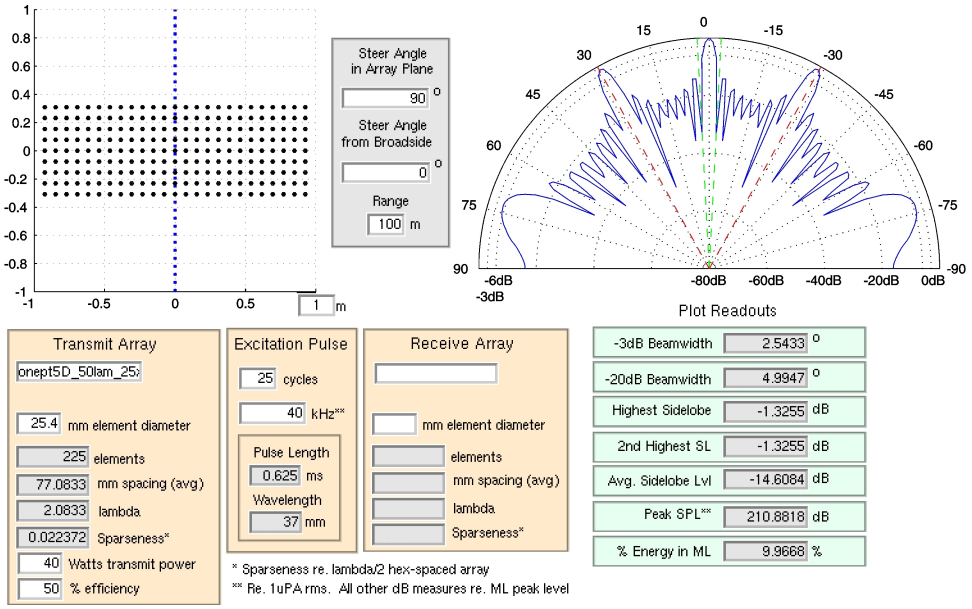


Figure 11: 1.5D array off-axis beam pattern (upper-left) shows some capability to focus in the vertical direction

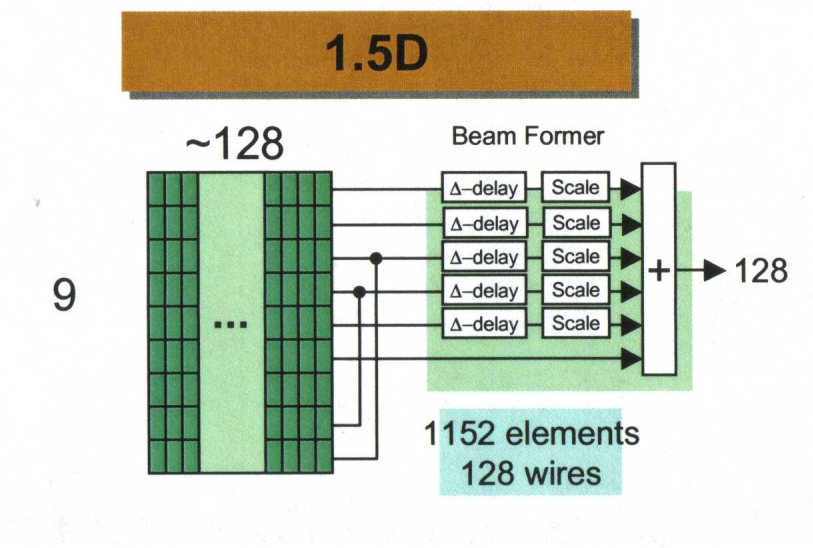


Figure 12: Using symmetry with 1.5D array to reduce calculations needed (Courtesy of K. Erikson et al. [9])

1 dimension, spacing at anything greater than $\lambda/2$ can cause high sidelobes to appear in the beam patterns in both directions. A 2D array at the same spacing as the 1.5D array previously considered would require 625 elements, while a $\lambda/2$-spaced array of the same size would require over 10,000 as shown in Figure 13.

3.3.1 Linearly Spaced Array Simulation Results

Here, the presence of the high sidelobes seen in the previous 1D and 1.5D cases is practically eliminated due to the smaller spacing of the elements. However, there are 400 times the number of elements used in the initial 1D case and 44 times as many as the 1.5D case. Computation times scale proportionately with element numbers, and can become cumbersome.

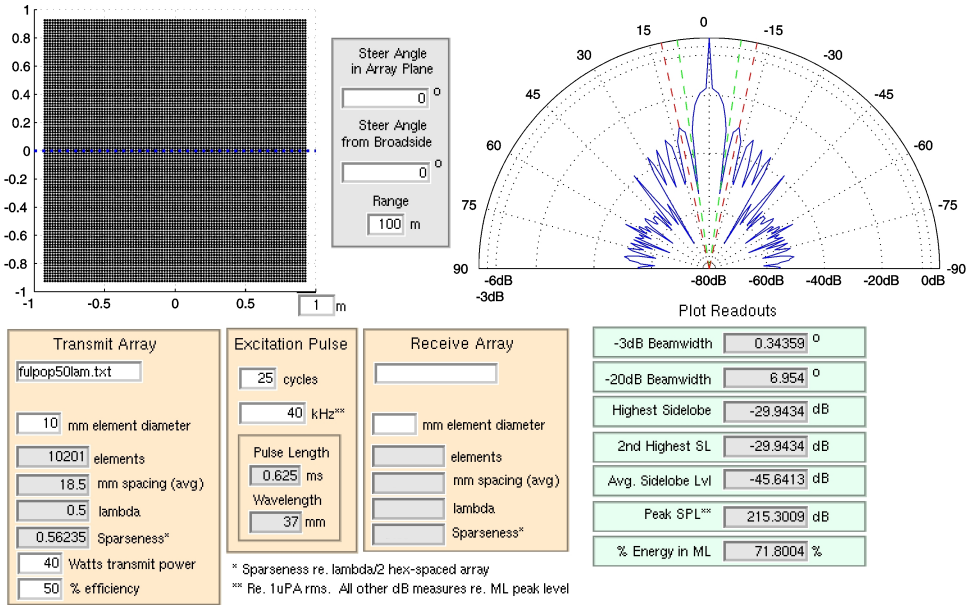


Figure 13: Fully-populated 2D array at $\lambda/2$ spacing and its resultant simulated beam pattern

Thus, while the 2D linear array is straightforward, for decades the attempt has been made to achieve comparable or even more desirable array performance with a more sparse element distribution.

3.3.2 Diagonally Optimized Array Pair Results

One such solution was shown by Nikolov and Jensen when proposing "effective aperture" methods for array design [6]. This concept takes advantage of the fact that a pulse-echo acoustic imaging system can have a separate transmit and receive array, which when used together can be thought of as one "effective" array aperture. By using different element spacing for the transmit and receive arrays, the combined effective aperture can be given $\lambda/2$ spacing without the need for either array to have such fine spacing alone. This technique is referred to as the "rectangular vernier approximation." To demonstrate this principle, consider the 2D rectangular array shown in Figure 14. This demonstrates the pulse echo response of a 2D transmit and receive array with 1089 elements each. The aperture is 16λ wide in both directions. The beam pattern shown has many ideal properties such as minimal sidelobes and 90% of the pulse-echo energy in the main beam. Note that the beam pattern shown is the pulse-echo response using the same array for transmit and receive. By using Nikolov and Jensen's proposed vernier approximation the number of transmit and receive elements can be reduced to 121 and 81 respectively as shown in Figure 15. Here, the $\lambda/2$ transmitter spacing has been increased to $3\lambda/2$ while the receiver array spacing is increased to $4\lambda/2$. The main beam width for this array pair is little changed from the fully-populated case using only 1/10th of the total number of elements. The main drawback seen is an increased sidelobe level, which in turn leads to a decreased percentage of total energy in the main lobe. This affects the response even in the much smaller interval of $\pm 30°$, as the peak sidelobes at -19dB occur at roughly these angles. Nikolov and Jensen go on to show that the effective aperture in for this vernier array maintain $\lambda/2$ spacing along rows and columns, but not diagonally, causing the high grating lobes seen. By adding additional receiving elements along the diagonals as shown in Figure 16, the high sidelobes at these angles are eliminated, producing a beam response with a peak sidelobe of -26dB, with 61% of the energy in the main beam. Furthermore, this diagonally optimized array is an excellent approximation to the $\lambda/2$ spaced array over the interval $\pm 30°$ while still using only 12% of the elements used in the dense array.

The beam patterns in Figures 15 and 16 are similar to those shown by Nikolov and Jensen in [6], though they show only the interval $\pm 30°$. This provided early

assurance, however, that the array simulations performed were being carried out in a manner consistent with other Field II-generated results.

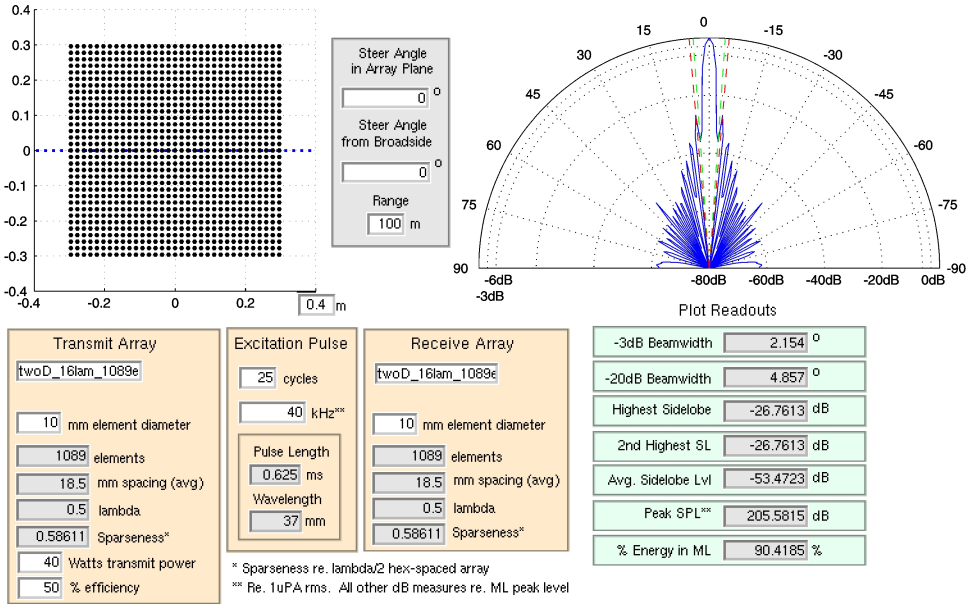


Figure 14: Fully-populated 16λ aperture with $\lambda/2$ spacing

3.3.3 Potential Simplification of 2D Rectangular Aperture Array Signal Processing

Aside from the proposed techniques for minimizing the number of elements needed for a particular array aperture pair, other methods can be used to simplify the signal processing for a fully-populated array. Figure 17 shows how an array of 16,000 elements can be steered by separating the processing into rows and columns. Delay values for transmission and reception need only be calculated for each row and column (256 total) rather than each element. The primary drawback to this scheme is some degree of loss of focusing control along the diagonal of the array. Nonetheless, the reduction in independent signals from n^2 to $2n$ is very significant.

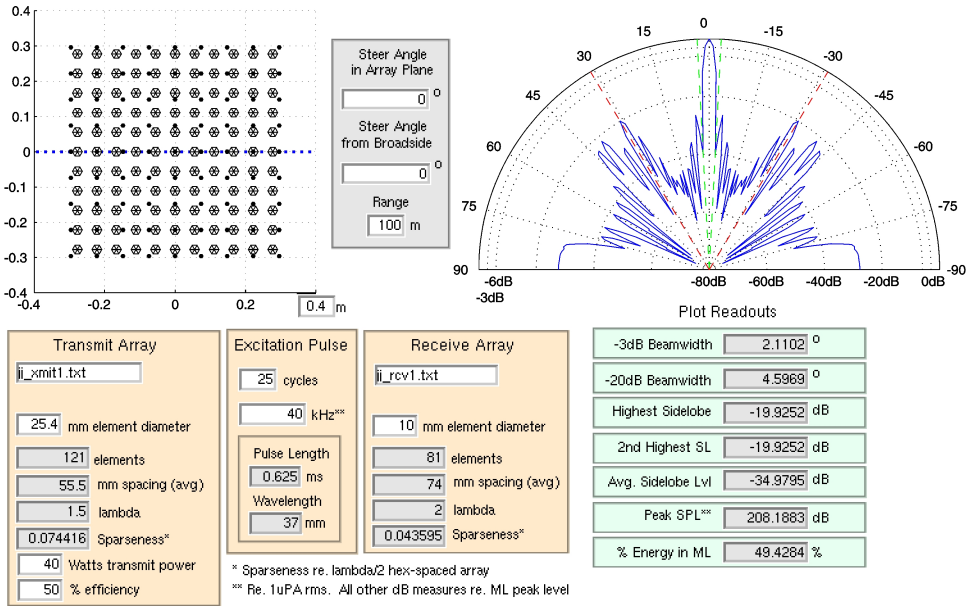


Figure 15: Vernier array shows higher sidelobes, similar beamwidth to $\lambda/2$-spaced array

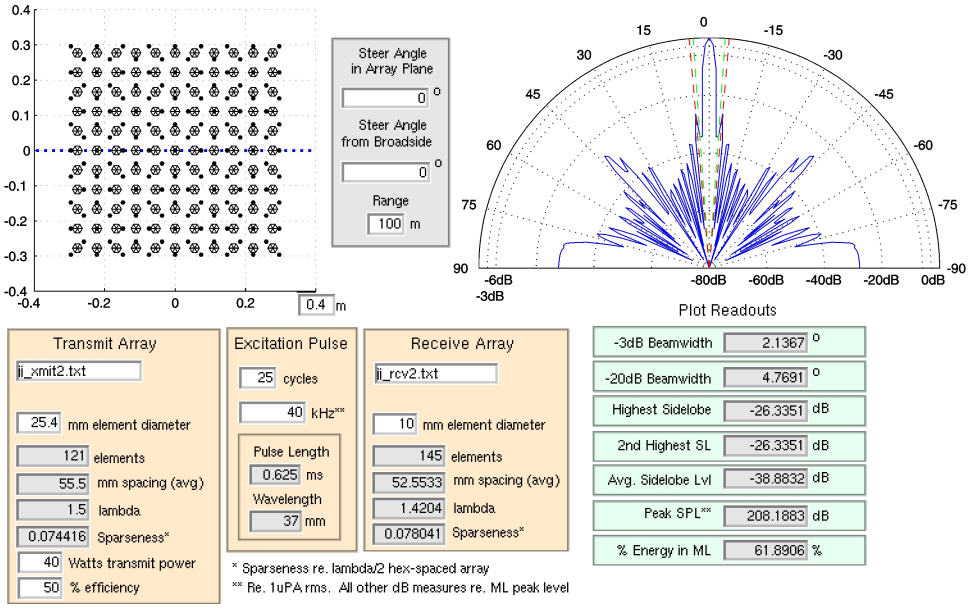


Figure 16: Diagonally-optimized Vernier array has lower sidelobes than unoptimized Vernier array

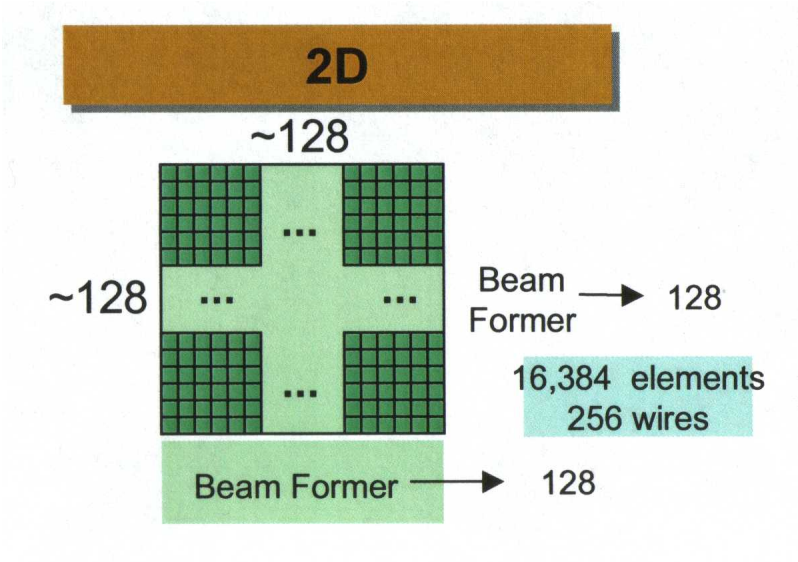


Figure 17: Independent X and Y beamformers used to simplify signal processing for 2D array (Courtesy of K. Erikson et al. [9])

3.4 2D Circular Aperture Array

While the rectangular linearly-spaced 2D array is a logical extension of the 1D array, its lack of radial symmetry makes it better at steering in some directions than in others. For example the diagonally optimized vernier array discussed previously and shown in Figure 16 has much lower sidelobe levels but a wider beamwidth at -3dB and -20dB when analyzed along the array diagonal as shown in Figure 18. An array whose elements are roughly symmetric about a fixed point of rotation will have a uniform ability to focus at all angles in the array plane. The following sections describe two arrays that accomplish this objective to varying degrees of success.

3.4.1 Hexagonally-Spaced Array Simulation Results

Besides being the most dense way to pack circular objects, a hexagonal grid pattern benefits from some degree of radial symmetry. The array shown in Figure 19, for example, maintains this same beam shape along diagonals at 60° and 120°, rather than only at 90° as is the case for the rectangular array. Additionally, the regular spacing of such a pattern (even in Cartesian coordinates) makes it relatively practical

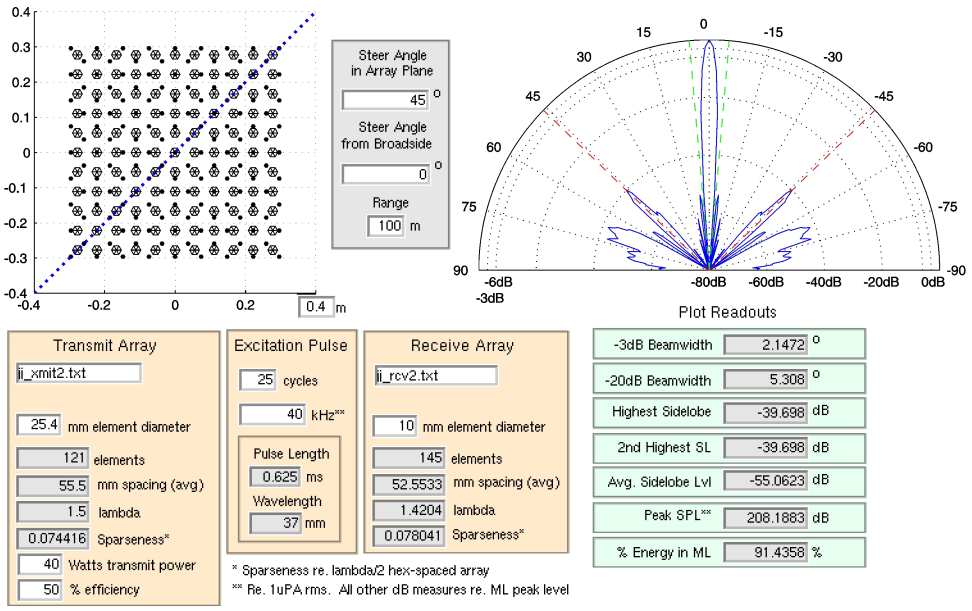


Figure 18: Beam pattern of diagonally-optimized Vernier array at 45° differs from that at 0° (shown in Figure 3.3.2)

to manufacture. Conversely, this same regular pattern at approximately $7\lambda/2$ spacing causes very high sidelobes in the beam pattern only 2dB below the response at the focal angle. Furthermore, the beam shape at angles other than multiples of 60 suffers from the same degradation seen in the rectangular array case. An angle of 30°, for example, yields the pattern shown in Figure 20 with 2 sets of sidelobes at levels within 3dB of the main lobe.

3.4.2 Concentric Rings Simulation Results

Increased radial symmetry is obtained by abandoning regular Cartesian spacing in favor of polar coordinates. The array shown in 21 consists of 4 concentric circular rings of elements with roughly the same sized aperture and number of elements as the hexagonal array considered previously. While the main beamwidth and peak sidelobe levels are improved over the hexagonal array, the average sidelobe level is greater, leading also to the reduced percentage of energy concentrated in the main lobe. However, the beam profile along a 30° diagonal, shown in Figure 22, is comparable to the

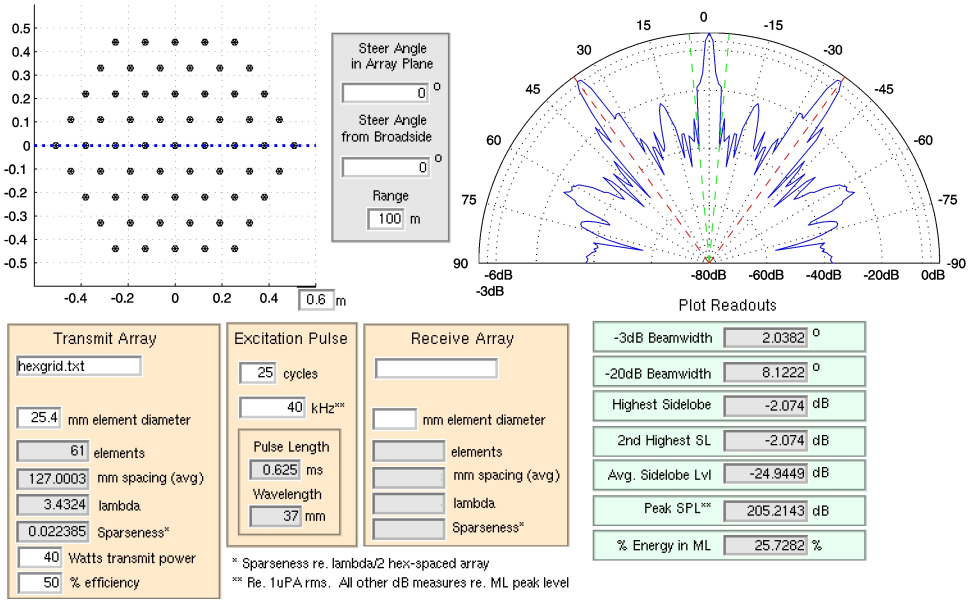


Figure 19: Hexagonally-spaced hexagon array and resultant beam pattern

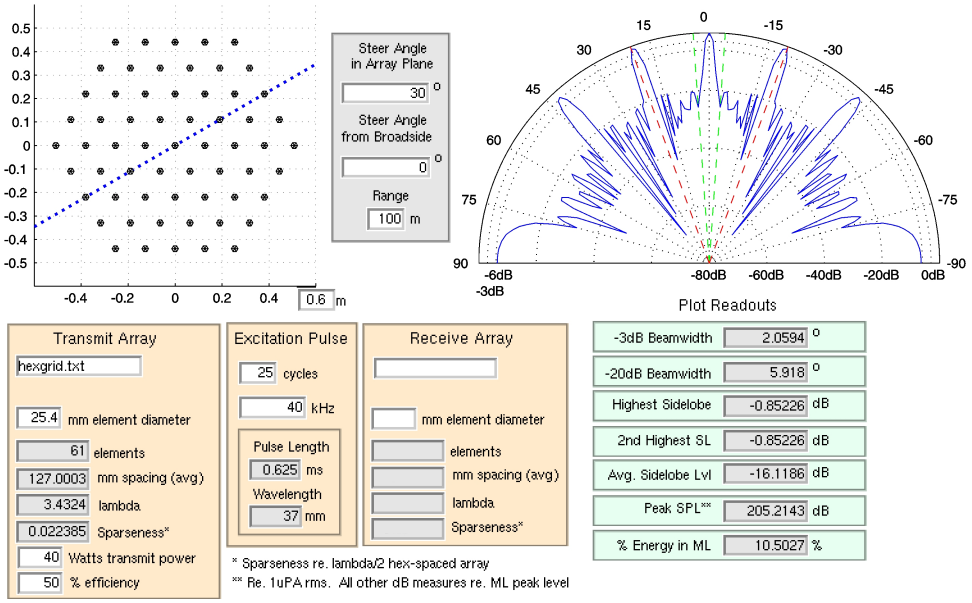


Figure 20: Hexagonal array beam pattern has higher and more sidelobes along a 30° diagonal

horizontal profile. The beam pattern of this array exhibits a high degree of radial symmetry, as the array itself does.

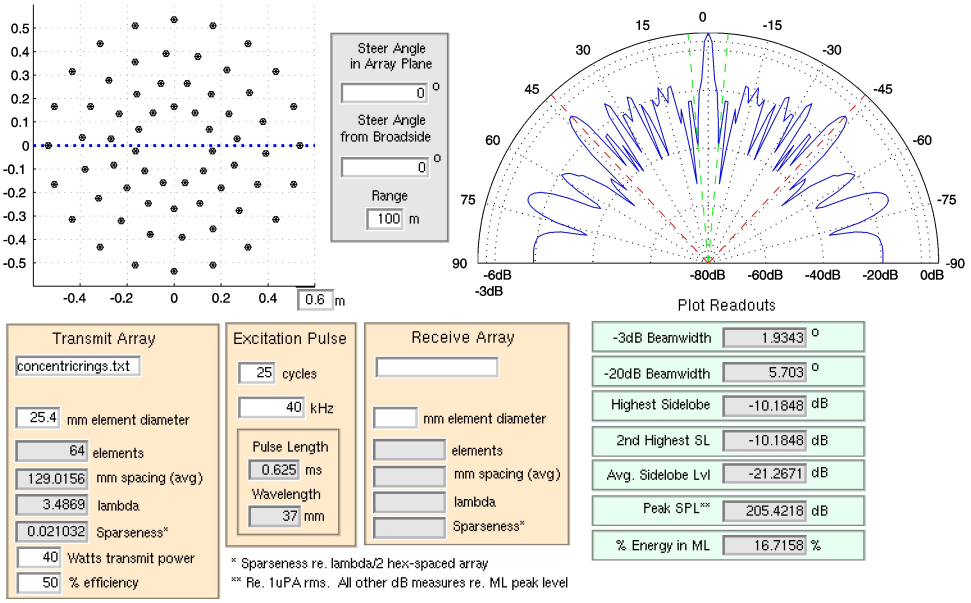


Figure 21: Array of concentric rings of elements and the resultant beam pattern

4 Optimization Methods

Array Pair Degrees of Freedom	
Number of Transmitters (m)	1
2D Coordinates for each Transmitter	$2m$
Number of Receiving Elements (n)	1
2D Coordinates for each Receiver	$2n$
Total	$2(m+n+1)$

Table 6: Degrees of freedom in optimization problem

Finding the best configuration of transmit and receive acoustic arrays for a particular set of circumstances is an inherently difficult task. Table 6 shows that an

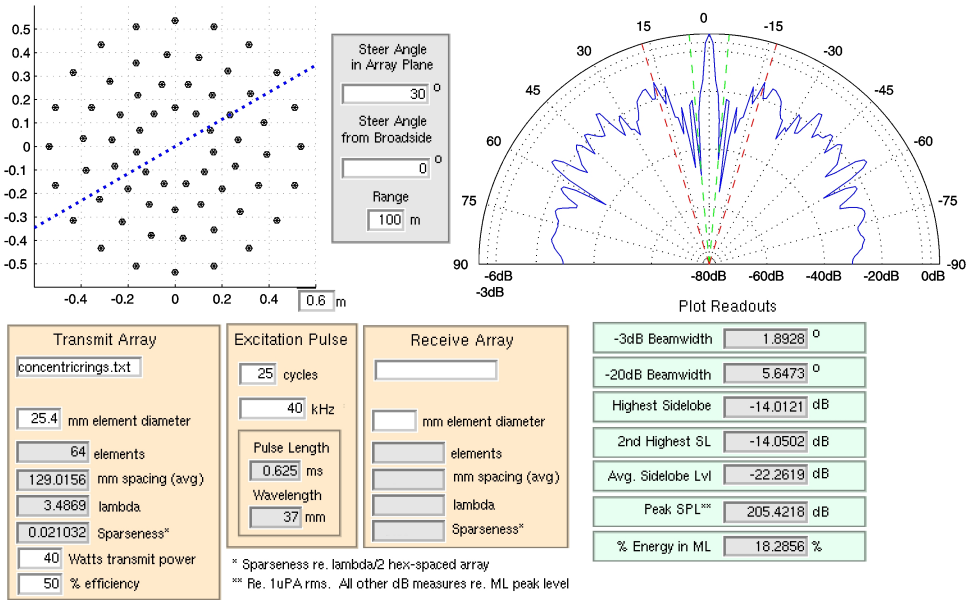


Figure 22: Concentric ring array beam pattern at 30° resembles that at 0°

array pair with m transmit elements and n receive elements can be thought of as a system of $2(m+n+1)$ independent variables. For the concentric ring array considered in Section 3.4.2, this amounts to a 258-dimensional problem. Arriving at a single solution in such high-dimensional space is exceedingly cumbersome under the most favorable circumstances, and for most practical situations impossible to achieve in any reasonable amount of time. For the concentric ring array, if each variable were restricted to one of only 10 values, to search all possible arrays would require the consideration of 10^{256}different array pairs. A trillion computers, each able to analyze 1 trillion possibilities per second would still require over 10^{224} years to definitively find the optimum 64×64 element array pair using those restricted values. The realization of the immensity of such problems has gives rise to the notion that often a "good" solution will suffice. Indeed if a "good" solution can be found quickly and is a good approximation to the optimum solution, then perhaps there is little to be gained in seeking the one best solution.

Simple optimization schemes abound for finding fairly good solutions to such problems. A straightforward hill climb algorithm can find local minima in a cost function

within a few dozen incremental changes in each variable. Unfortunately, if a large number of local minima are present in the cost function, the optimized solution will be extremely dependent on the starting array configurations. Stochastic optimization techniques introduce controlled randomness into the hill climb process to avoid getting "stuck" in a sub-optimal local minimum solution. Two different such optimization methods were considered for application to acoustic array topologies. Genetic algorithms such as the one proposed by Weber, Peter, and Aakvaak [7] generate new array topologies by selecting portions of a number of existing topologies, combining them, and allowing for some random mutations. Trucco presents another array optimization scheme based on simulated annealing [8]. Here new arrays can be generated by repeatedly perturbing the elements of a pre-existing array individually. A similar technique was implemented by Hopperstad and Holm [4] using as a starting point the 25 element 1D array described in Section 2.1.1. For this thesis, a simulated annealing strategy for optimization was chosen over a comparable genetic algorithm because it offered a more incremental way of generating new arrays, which was deemed to be desirable for a straightforward implementation.

4.1 Simulated Annealing Method

Simulated annealing is designed to mimic the manner in which liquids freeze or metals recrystallize in the process of annealing. During annealing, a melt starts at a high temperature in a disordered state. As the it cools slowly, molecular orientation within the melt becomes more ordered until "frozen" at a zero temperature condition. The process can be thought of as an approach to a low energy state. In simulated annealing this energy state is replaced by a generic cost measure to be minimized. The random thermal motion of molecules is replaced with numerical perturbations of a number (usually many) of independent variables.

The simulated annealing algorithm developed for this thesis was patterned after the one described by Trucco. One of the key differences in adaptation is the allowance for an extremely flexible cost measure (described in Section 4.2). Also, the technique was applied to an array pair in the bistatic transmission case rather than just to a single array for monostatic use.

The first step in carrying out an array pair optimization using simulated annealing

is to determine how the system temperature will vary for each iteration, n. For simplicity, an exponentially decaying non-dimensional temperature parameter $T(n)$ is used, normalized from 1 to 0. This starts each simulation allowing a high degree of randomness on the first iteration, while allowing none on the last. For each iteration of simulated annealing, equation 1 is used to determine the system temperature value. N is the total number of iterations and K is a user-specified time constant representing roughly the number of iterations at which the system temperature should drop by a factor of e^{-1}. All simulations in this thesis are performed with $K = 1$.

$$T(n) = \exp(\frac{1-n}{K}) - \exp(\frac{1-N}{K}) \tag{1}$$

The optimization engine is invoked, which computes the initial system cost value for the original transmit/receive array pair. The Matlab code for this can be found in Appendix B. An "unconsidered" list is initialized which includes all transmit and receive elements. This assures that all elements will be perturbed while allowing for consideration in random order. Finally, the main optimization loop begins: A single element is chosen at random from the unconsidered list and subsequently removed from the list. User-defined probabilities influence whether the chosen element will split into 2 elements, move in a random direction, or be removed completely from the array (die). If the element is considered for being split into 2 elements, the old element remains at its current location while a second element is added at a random direction and distance. The new element is checked for overlap with other existing elements. If this occurs, an attempt is made to place the second element in a different random location. If after several attempts, there is no location found where a second element can be placed without overlapping with neighbors, the consideration for element splitting is aborted and the element is either considered for moving or death. If an element is being considered for moving, it is also checked for any possible element overlap. If no suitable move possibilities are found, the element is then considered for death. Regardless of the manner of perturbation, a new array is generated with the perturbation imposed. This new array is simulated with the array plotter and a corresponding beam pattern is computed. Since the perturbation only affects either the transmit or receive array, (not both) the unchanged array does not require re-simulation. (This reduces computation time by a factor of 2.) The previously com-

puted beam pattern for the unchanged array is multiplied with the new array beam pattern to yield the new pulse-echo beam pattern. If the optimization is being run with multiple steer angles or pulse frequencies, the array simulation is repeated at each combination of steer angle and pulse frequency, and the beam patterns derived for each. From these patterns a single cost parameter is derived for the new array pair and compared to the current cost number computed previously. These comparisons are shown in Equations 2, 3, and 4. If the new array pair has a lower cost than the previous pair, the array change is accepted and made permanent. If the new cost is higher, then the difference between the new and old costs becomes significant. A random number between 0 and 1 is chosen and multiplied with the cost difference. If this number is less than the current system temperature, then the array change is accepted. Otherwise, the perturbed array is rejected and both arrays remain the same.

$$C_{new} < C_{old} \qquad Accept \tag{2}$$

$$(C_{new} - C_{old}) \cdot Rand < T_{cur} \qquad Accept \tag{3}$$

$$(C_{new} - C_{old}) \cdot Rand > T_{cur} \qquad Reject \tag{4}$$

Following the cost comparisons the unconsidered list is checked for non-zero length. If the list is not empty, a new element is chosen at random from the list and perturbed using the same procedure just described. If, however, the unconsidered list is empty, the iteration number is incremented and compared to the total number of iterations. The optimization terminates if the new iteration value is greater than the requested number N. If it is less than N, the system temperature is updated and the unconsidered list is repopulated with all transmit and receive elements (including any new elements resulting from splits, minus any elements that died). An element is chosen from this list, it is perturbed as before, and the optimization continues as previously described. Figure 23 illustrates the flow of the optimization procedures.

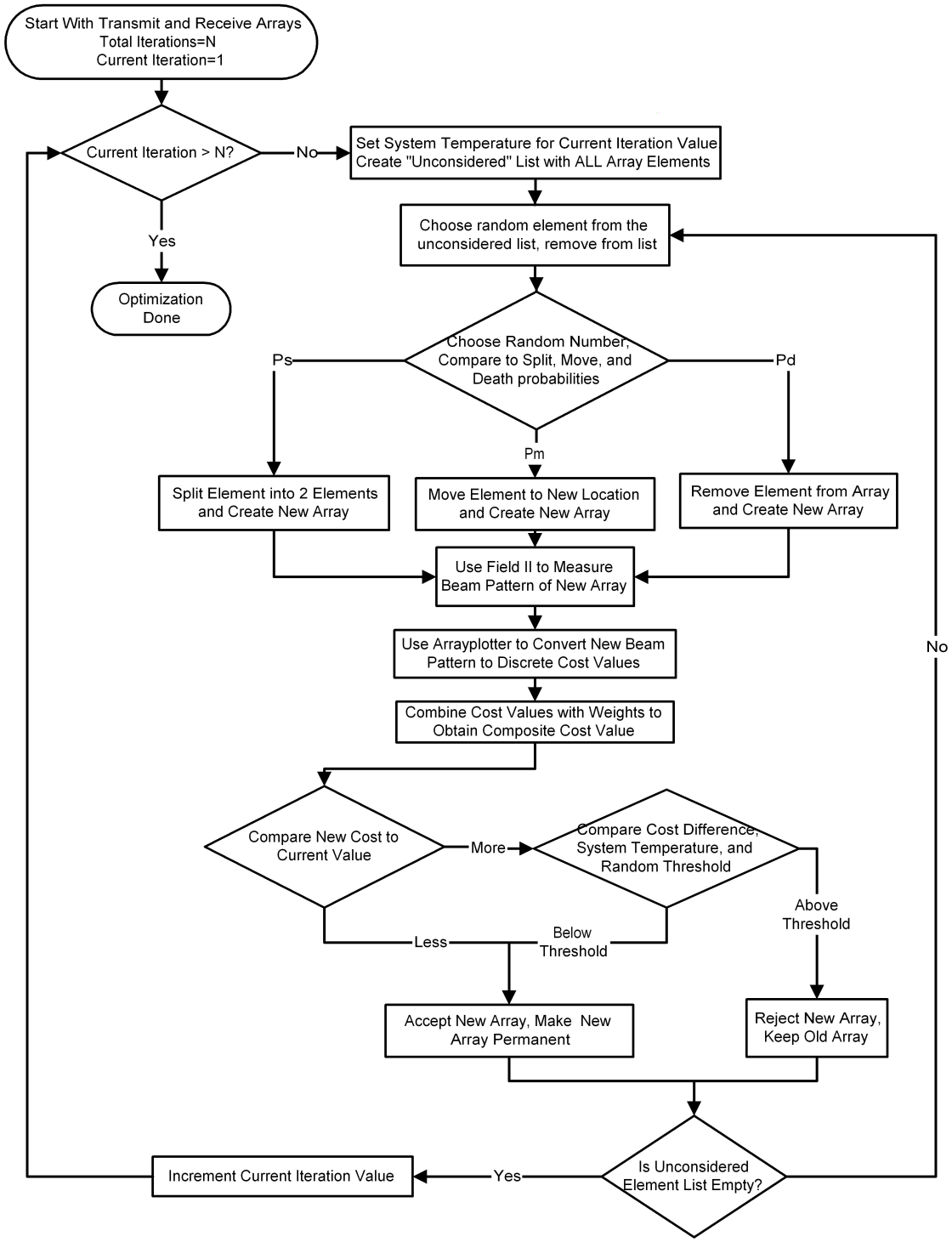


Figure 23: Optimization flowchart

4.2 Cost Functions

At the heart of each iteration of the simulated annealing algorithm is a measure of system "cost." For a cooling material during annealing this cost is the stored energy lost as the particles of the material move to a lower energy state. In general this system cost measure can be any undesirable system property to be minimized. By identifying many different measures of array "goodness" and combining them, an all-inclusive cost function can be created. Furthermore, by weighting the different cost measures within the composite cost value, the cost function can be custom-tailored to meet the requirements for varying scenarios.

4.2.1 Number of Elements

Two of the most basic measures of cost are the number of transmit array elements and the number of receive array elements. These are mostly practical considerations. Of particular note is the separation of the two cost measures. In a bistatic imaging situation, there are generally greater hardware requirements for a transmitting element than a receiving element. Often there are individual power amplifiers for each, which makes them have a higher monetary cost than receiving elements, and often they are larger than receivers, giving them a higher spatial cost than receiving elements. Such factors should be carefully considered before setting the weights for these cost functions.

4.2.2 Peak Sound Pressure Level (SPL)

A counteracting "cost" measure to the number of transmit elements is the sound pressure level emitted by the transmitting array (SPL). This pressure level is calculated at one meter from the array surface, and as it is a desirable quality, this measure contributes a higher cost as it decreases. To guarantee that an acoustic imaging system will be able to resolve objects a certain distance away, the transmit array as a whole must meet minimum power requirements. The weight for this cost measure can be set in accordance with those requirements.

4.2.3 Beamwidths

As presented at the beginning of this thesis, the beamwidth of the main lobe is the key determining factor in the resolution of the imaging system as a whole. Separate cost measures are provided for controlling the -3dB beamwidth and -20dB beamwidth.

4.2.4 Sidelobe Levels

Three different cost functions measure sidelobe levels. The first two track the peak sidelobe level and the 2nd highest sidelobe level. It was thought that since high sidelobes generally come in pairs, that tracking changes in both would add redundancy and accuracy to this measure. Most practical uses of array optimization give these costs significant weight. The third sidelobe cost allows the user to assign a relative weight to the average sidelobe level.

4.2.5 Energy Ratio, Main Lobe:Total

If a user desires an array pair that will perform well in a noisy environment, the main lobe energy ratio cost function is of importance. As with the Peak SPL cost function, this "cost" function contributes the least cost to the composite when maximized. Consider the value of this function in an isotropic noise scenario, for example. If the ratio is below 50%, echoes from a target at the array steer angle will not be distinguishable from ambient noise. This is because the echo signal from a target located at the array steer angle will contribute less to the total signal received than the integrated noise received at all other angles. In other words, the signal to noise ratio of the imaging system drops below 1. Thus, the appropriate weighting for this cost measure depends greatly on the environment in which the arrays will be used.

4.2.6 Sidelobe Angular Proximity

Transmit and receive arrays with sidelobes at the same angles naturally produce a pulse-echo response with high sidelobes at those angles. An additional cost function was added to track the angular proximity of the transmit array sidelobes and the receive array sidelobes. Again, this function is costly when the angular sidelobe proximity is small (often zero). Initially this cost measure was seen as uniquely

indicative of certain undesirable system conditions. However, after some use it became apparent that the sidelobe proximity indications were merely a subset of the sidelobe level cost function. While it has been implemented and is useful to some degree, this cost function was rarely used.

4.2.7 Composite Cost Measure

All 10 cost functions described above must be combined into a single cost measure for use with the simulated annealing algorithm. First each is offset and scaled so that most reasonable cost function values will yield a number on the interval [0,1]. Each cost number is then multiplied by the a user-specified weight, and the sum of the weighted cost functions is used as the composite cost value corresponding to a distinct transmit/receive array pair as shown in 5.

$$C_{composite} = w_1 \cdot C_{BW_{-3dB}} + w_2 \cdot C_{BW_{-20dB}} + w_3 \cdot C_{SL_1} + w_4 \cdot C_{SL_2} + w_5 \cdot C_{SL_{avg}} + \quad (5)$$
$$w_6 \cdot C_{ML_{\%E}} + w_7 \cdot C_{SPL} + w_8 \cdot C_{N_{xmit}} + w_9 \cdot C_{N_{rcv}} + w_{10} \cdot C_{SL_{angprox}}$$

When multiple steer angles or pulse frequencies are used, the maximum normalized cost value is used for each individual cost measure. Thus, if sidelobes are high and beamwidth small at a 45° steer angle, and sidelobes are low but beamwidth large at broadside, the composite cost function will reflect high sidelobes and large beamwidth. In this manner a cost ceiling is set for all steer angles and frequencies, and cost functions for all are minimized simultaneously.

5 Optimization Results

5.1 1D Optimization Results

1D optimizations were run in Matlab 5.3 in Linux on an Intel Pentium III 500 MHz PC. Each is 20 iterations and required approximately one hour to run. All optimized 1D arrays shown start with the array pair shown in Figure 24. Both transmit and receive arrays are identical in dimensions to the array first shown in Section 2.2.1.

The receiving array was displaced horizontally by λ to avoid an initial condition with overlapping elements. Figures in the next two sections show the receiving array displaced along the vertical axis as well, but this is for ease of array visualization and comparison. Optimizations were done with all elements on the horizontal axis.

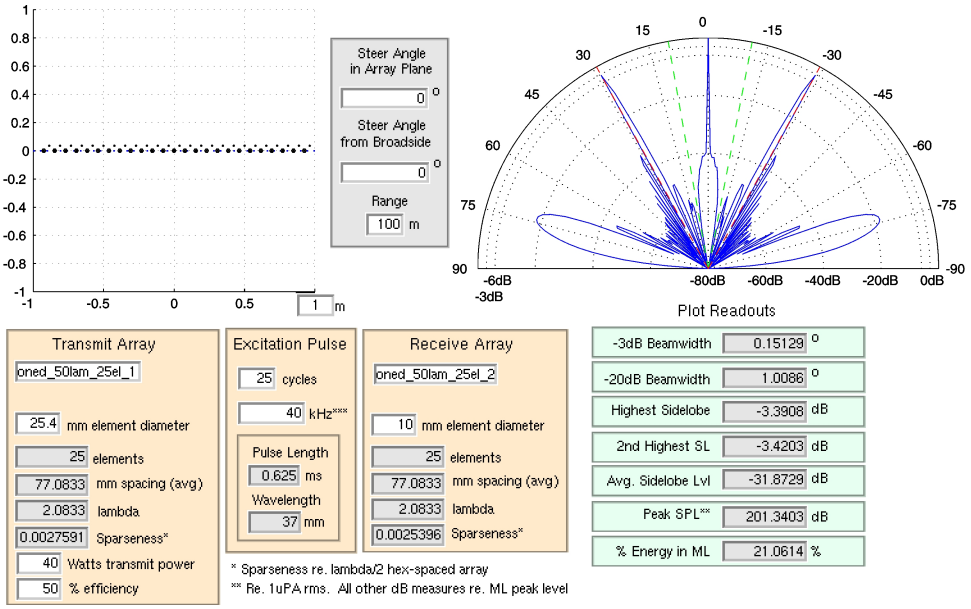


Figure 24: Starting array pair for 1D optimizations

5.1.1 Beamwidth

Figure 25 shows the Array Plotter interface augmented with optimization information. Here, the optimization was restricted to 1 dimension and consisted of 20 iterations, as shown in the bar at the top of the figure. The cost function weights used for the optimization are shown beside the beam plot readouts at the lower-right. Thus, Figure 25 shows that the array which started at a -3dB beamwidth of 0.15° and a -20dB beamwidth of 1.01° now has beamwidths of 0.07° and 0.49° respectively. The objective of reducing beamwidth was met. Upon inspection of the optimized arrays it was noted that element density on the outer portions of the arrays is greater than that in the middle. Concentrating elements at aperture extremities is a proven method

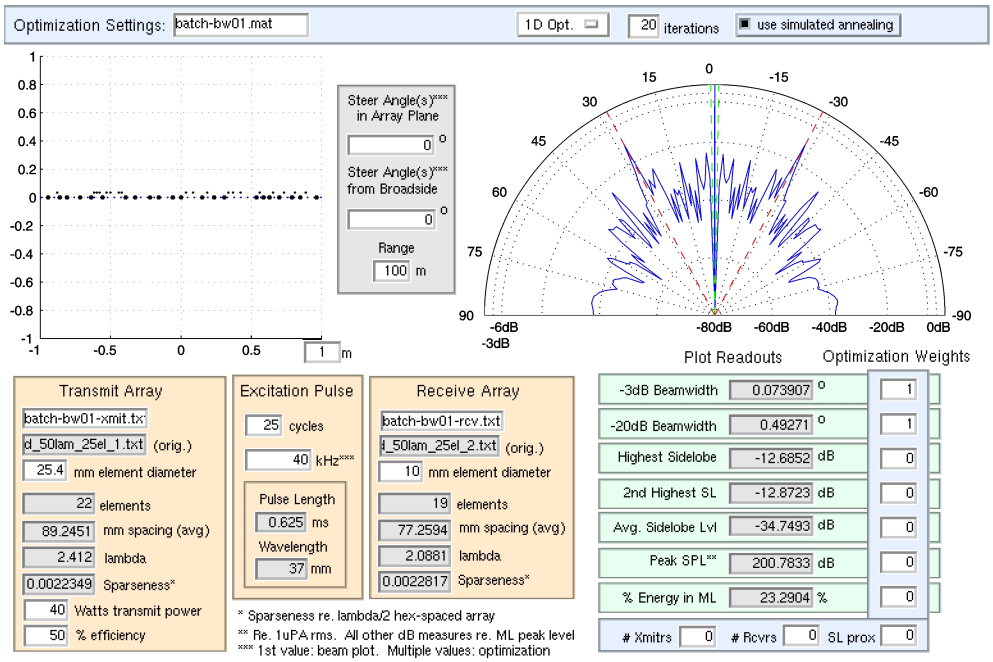


Figure 25: Array pair optimized for minimum beamwidth shows smaller beamwidth than unoptimized case (Figure 24)

of reducing beamwidth, a method inadvertently used by the simulated annealing algorithm due to the chosen cost function weights.

5.1.2 Peak Sidelobe Level

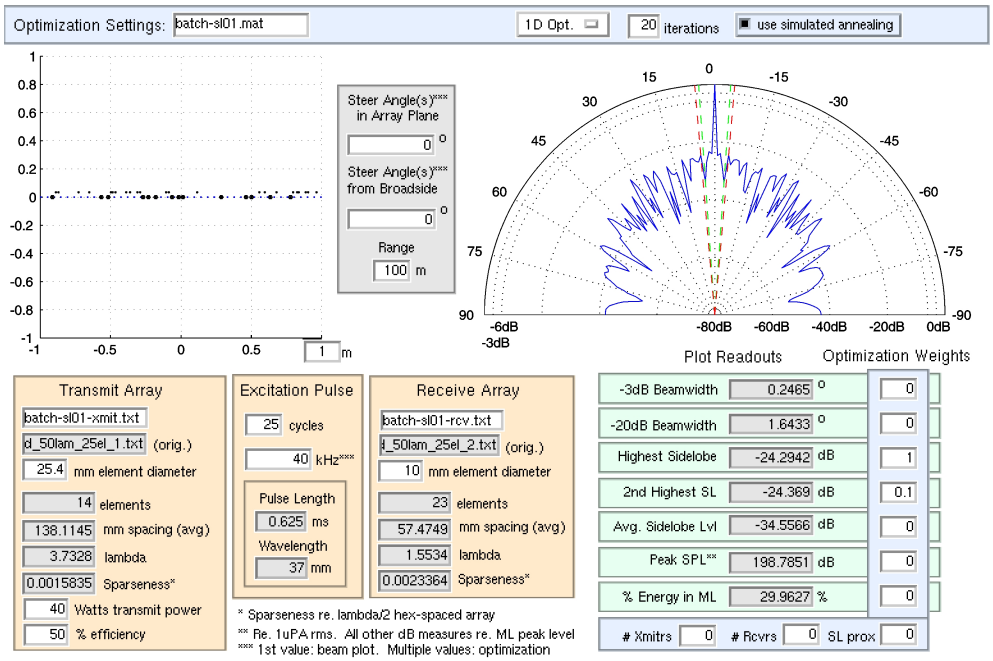


Figure 26: Array pair optimized for minimized peak sidelobe levels

Beamwidth measures for the optimized array in Figure 26 are worse than the original array pair, while peak sidelobe levels dropped over 20dB from the original arrays. This is as expected, since the only nonzero cost weights used were for the highest and 2nd highest sidelobe levels. In the beam plot shown, sidelobe levels are relatively flat outside the main lobe. This is the result of the fact that the peak sidelobe level cost function makes no distinctions in beam patterns below the peak sidelobe.

By comparison, in [4], the best known configuration of 25 elements on the same sized aperture resulted in a peak sidelobe level of -12.36dB. If this array were used as a transmit array and a receive array, the overall response would have a peak sidelobe level of -24.72dB. This is only slightly lower than the results shown in Figure

26 achieved in only 20 iterations. The elements in [4] were confined to a grid of $\lambda/2$ spacing, so that problem is indeed more restricted. The optimization results, however, are comparable, and this provided assurance that the optimization scheme developed for this thesis can perform comparably to other accepted methods.

5.1.3 Average Sidelobe Level

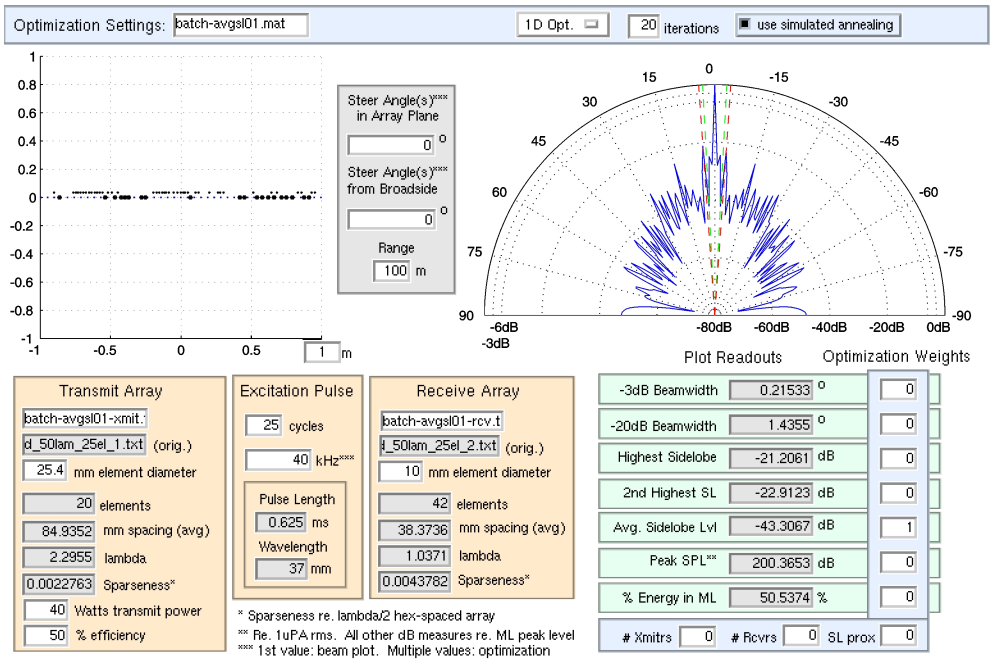


Figure 27: Array pair optimized for minimum average sidelobe level

The peak sidelobe level of the array pair in Figure 27 is indeed 3dB higher than the previous array which was optimized for that cost function. However this array pair, optimized for low average sidelobe level only, improves 11.4dB on the original array, and almost 20dB on the array optimized for peak sidelobe level. Of note with this array pair are the clumps of relatively regularly spaced elements. From the original fixed spacing of 2λ, the transmit array has increased that value to 2.3λ while the receiving array has reduced the spacing to λ. The different regular spacing for the transmit and receive arrays is the technique used to create vernier arrays with small effective spacing. This concept was presented in Section 3.3.2.

5.1.4 Percent Energy in Main Lobe

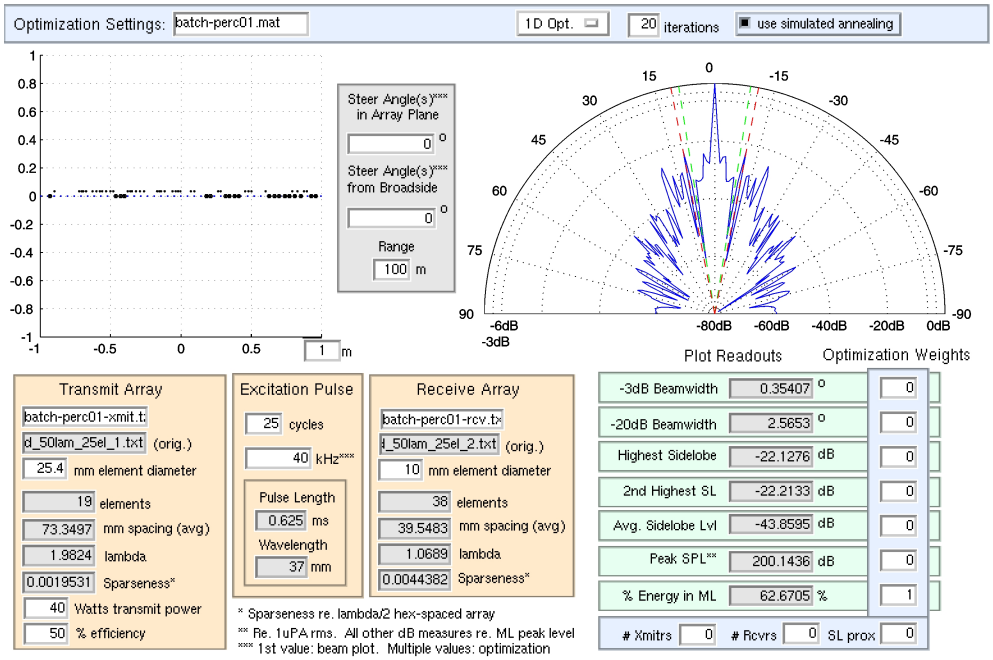


Figure 28: Array pair optimized for high main lobe energy ratio

The peak and average sidelobe calculations for the array pair in Figure 28 are virtually identical to those seen in the previous array pair optimized for average sidelobe level. The element numbers for transmit and receive are similar as well. The key difference lies in the relationship between beamwidth and main lobe energy ratio. Factors controlling the computed value for this energy ratio are, 1) the average sidelobe level, 2) the average main lobe level, and 3) the width of the main lobe. Thus, the optimization of the main lobe to total energy ratio attempts to lower the average sidelobe level with respect to the average main lobe level and widen the main lobe itself. The optimization in Figure 28 mimics the average sidelobe reduction of Figure 27 while adding an increase in main beam width. These two changes allowed this optimized array pair to achieve a 40% increase in main lobe energy ratio over the original array pair.

5.1.5 Number of Elements

Figures 29, 30, and 31 show the difference in array pair optimization when using no element costs, high element costs, and a 10:1 transmit/receive element cost ratio respectively. The final element counts for the three optimized array pairs are 20/20, 15/19, and 13/29. While in all three cases the -20dB beamwidth and the highest sidelobe level (the two other cost functions used in optimization) are significant improvements over the original array pair, the subtle differences in their beam patterns give insight into how element weighting affects optimization.

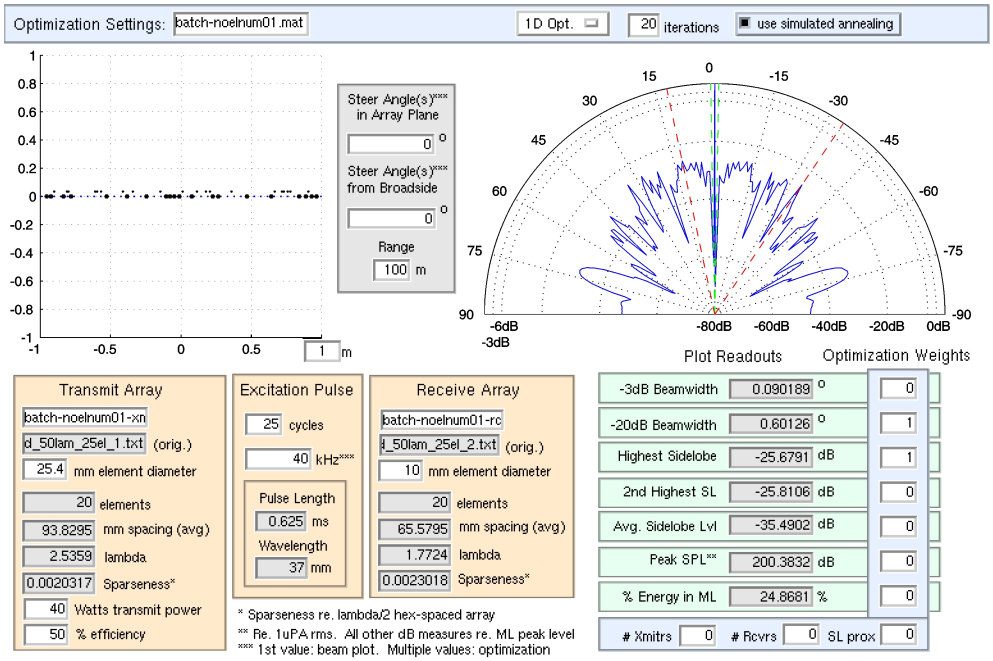


Figure 29: Array pair optimized for small beamwidth and low sidelobes without element costs

With no element costs, 40 elements total allowed the array pair in Figure 29 to have the lowest peak sidelobe level of the three optimizations, -25.7dB.

The array pair in Figure 30 has a peak sidelobe level 2dB higher than the array pair with no element cost, and a -20dB beamwidth 0.12° higher than the array pair with ratioed element costs. The smaller footprint of only 34 elements total accounts for the somewhat lesser performance.

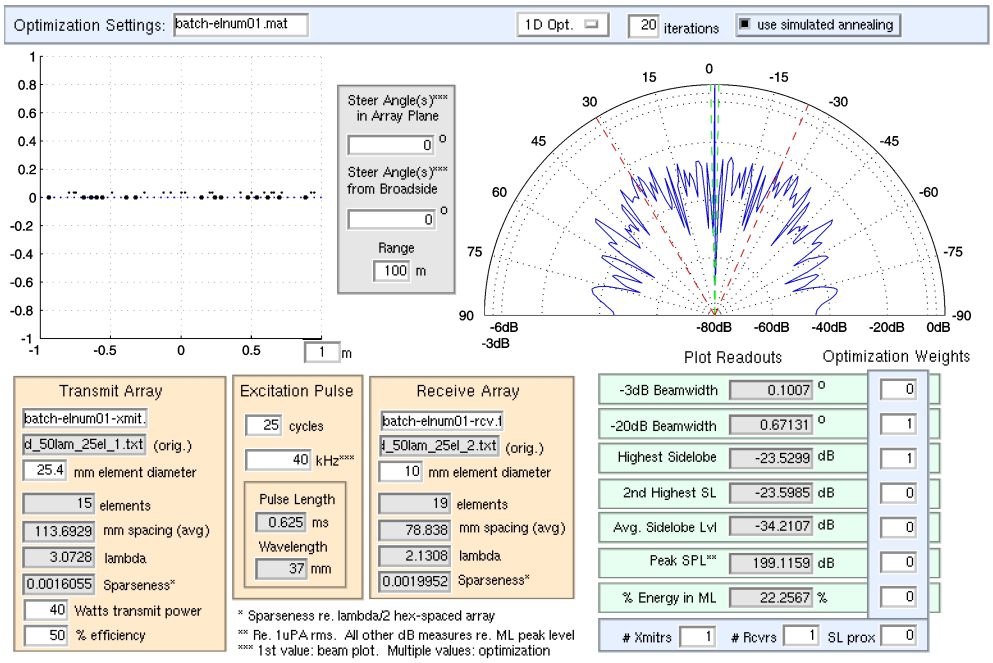


Figure 30: Array pair optimized for small beamwidth and low sidelobes with high, equal element costs uses fewer elements, but reaches a result with slightly larger beamwidth and sidelobes than the case with no element costs

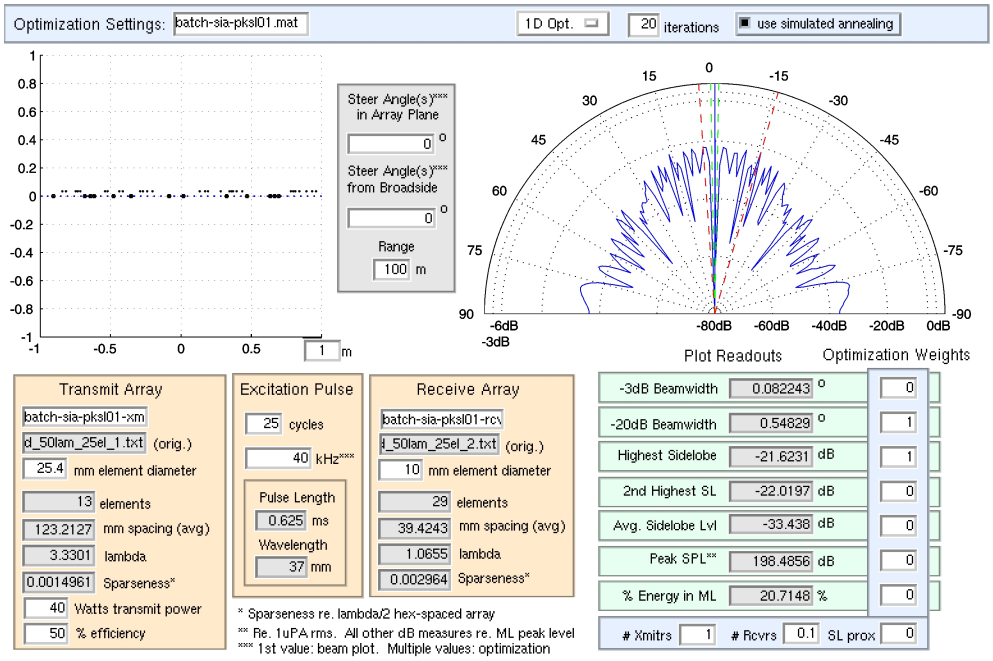


Figure 31: Array pair optimized for small beamwidth and low sidelobes with 10:1 transmit/receive element cost ratio uses roughly the same total number of elements, redistributed 2:1 toward the receive array, and achieves slightly smaller bandwidth but higher sidelobes than the case with no element costs

Figure 31 shows that with 42 elements total, a -20dB beamwidth 0.05° smaller than the case with no element cost can be obtained, with a 4dB increase in peak sidelobe level over the same case. This shows that, considering arrays pairs with roughly the same total number of elements, a pair with a higher percentage of small receiving elements will favor a narrow beamwidth, while an even transmit/receive element distribution favors array configurations with low peak sidelobes.

5.1.6 Summary

Tables 7 and 8 summarize the results obtained performing array pair optimizations using the cost weights mentioned previously. In all cases, 20 iterations of simulated annealing can create an array pair customized for a specific feature from a simple, linearly-spaced, 1D array pair.

	Figure Number:							
Cost Functions:	25	26	27	28	29	30	31	34
-3dB Beamwidth	1	0	0	0	0	0	0	0
-20dB Beamwidth	1	0	0	0	1	1	1	1
Highest SL	0	1	0	0	1	1	1	1
2nd Highest SL	0	0.1	0	0	0	0	0	0
Avg. SL	0	0	1	0	0	0	0	0
Transmit SPL	0	0	0	0	0	0	0	0
% Energy in ML	0	0	0	1	0	0	0	0
Transmit Elements	0	0	0	0	0	1	1	1
Receive Elements	0	0	0	0	0	1	0.1	0.1
Sidelobe Proximity	0	0	0	0	0	0	0	0

Table 7: Cost function weights used for optimization

5.2 Multiple Angle Optimization Results

5.2.1 Combined Beamwidth and Sidelobe Optimization

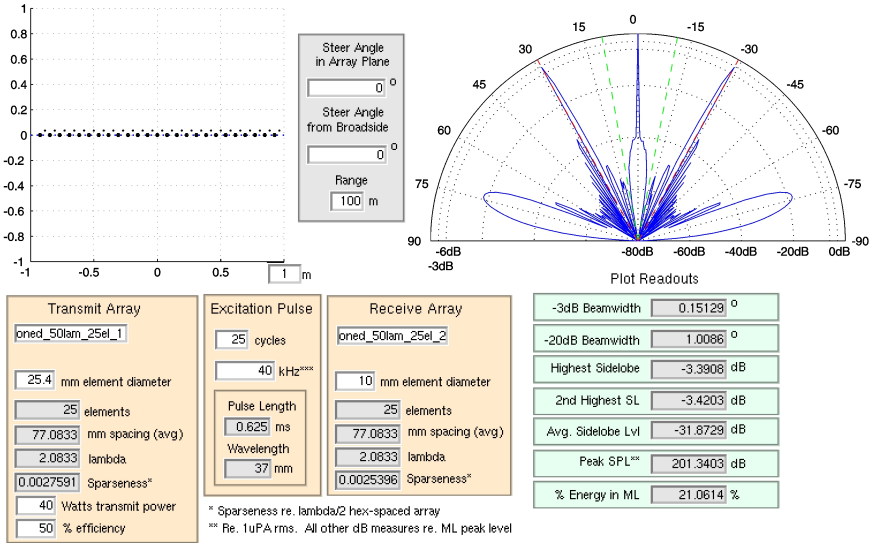
The following section demonstrates the ability of the array optimizer to find an optimum array while taking into account multiple steer angles and frequencies. Figure 32 shows the beam patterns for the starting array pair steered at 0° and 45° from

	Figure Number:							
Cost Functions:	24	25	26	27	28	29	30	31
-3dB Beamwidth (°)	0.15	0.07	0.25	0.22	0.35	0.09	0.10	0.08
-20dB Beamwidth (°)	1.01	0.49	1.64	1.43	2.56	0.60	0.67	0.55
Highest SL (dB)	-3.4	-12.7	-24.3	-21.2	-22.1	-25.7	-23.5	-21.6
Avg. SL (dB)	-31.9	-34.7	-34.6	-43.3	-43.9	-35.5	-34.2	-33.4
Transmit SPL (dB)	201.3	200.8	198.7	200.4	200.1	200.4	199.1	198.5
% Energy in ML	21.1	23.3	30.0	50.5	62.7	24.9	22.3	20.7
Transmit Elements	25	22	14	20	19	20	15	13
Receive Elements	25	19	23	42	38	20	19	29

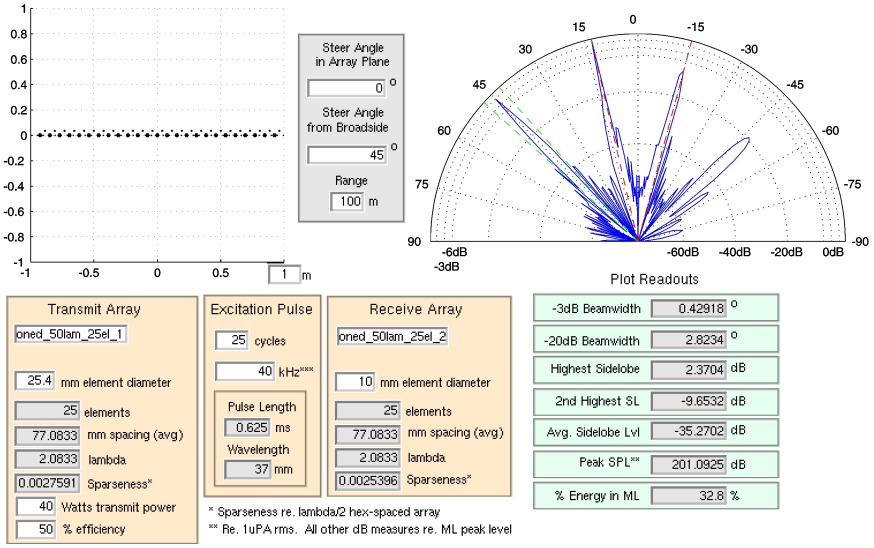
Table 8: Summary of 1D optimization results

broadside. The 45° steer plot shows a sidelobe 2dB higher than the main lobe! This array pair needs optimization to be practical at such a steer angle.

Figure 33 shows the same optimized array from Figure 31 and its beam pattern at 0° and 45° steer angles from broadside. The array optimization at 0° has greatly improved the array pair beam pattern even at a 45° steer angle. The peak sidelobe value is reduced from +2.3dB to -16.8dB, though the -20dB beamwidth increases from 2.8° to 4.0°. Figure 34 shows the array pair optimized at 0° and 45°. With this array pair, the highest sidelobe at the 45° steer angle is reduced further to -18.7dB and the -20dB beamwidth is 2.1°, an improvement on the original array pair at that steer angle. While beam parameters are improved at the 45° steer angle for this multiple-angle optimized array pair, the beam properties suffer somewhat when comparing the broadside performance to that of the single-angle optimized array pair. Here, while the multiple-angle array pair has identical performance at 0° and 45°, the single-angle optimized array pair has a 0.5° -20dB beamwidth and -21dB peak sidelobe level. Optimization over many different steer angles and/or pulse frequencies allows a user to sacrifice array pair performance under certain conditions for consistency over many different conditions.

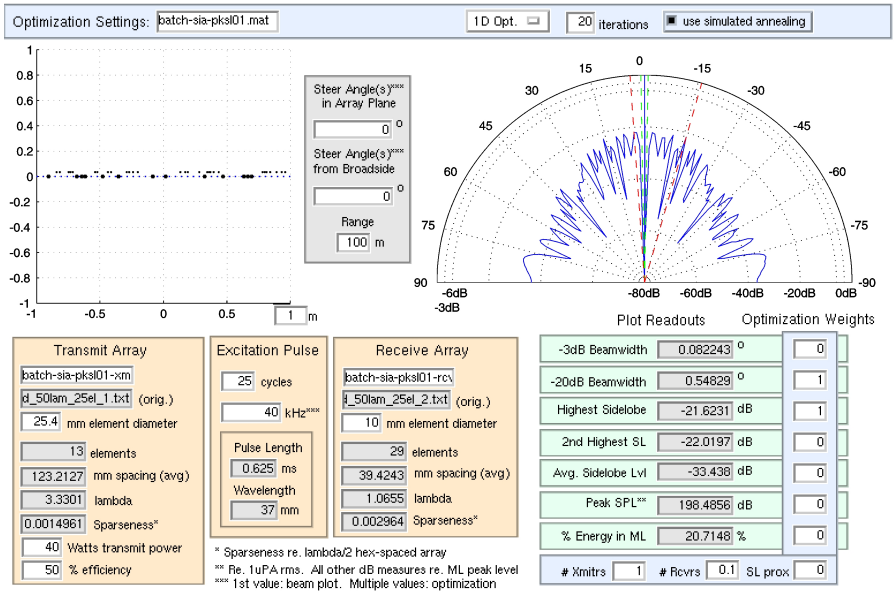


(a)

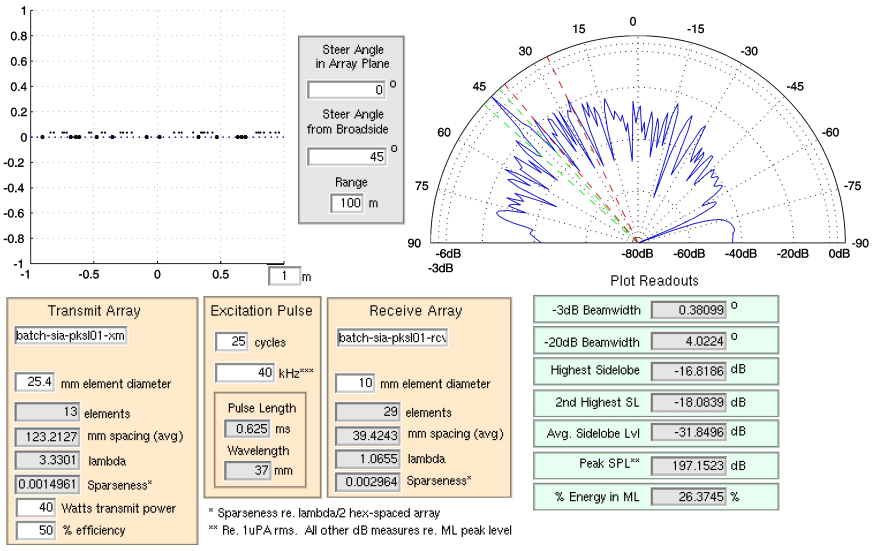


(b)

Figure 32: (a) 1D Array pair (no optimization) shows sidelobes -3dB lower than the main lobe at 0° (b) Same array pair with main lobe steered to +45° shows a peak sidelobe (at +13°) 2dB higher than the main lobe (at +45°)

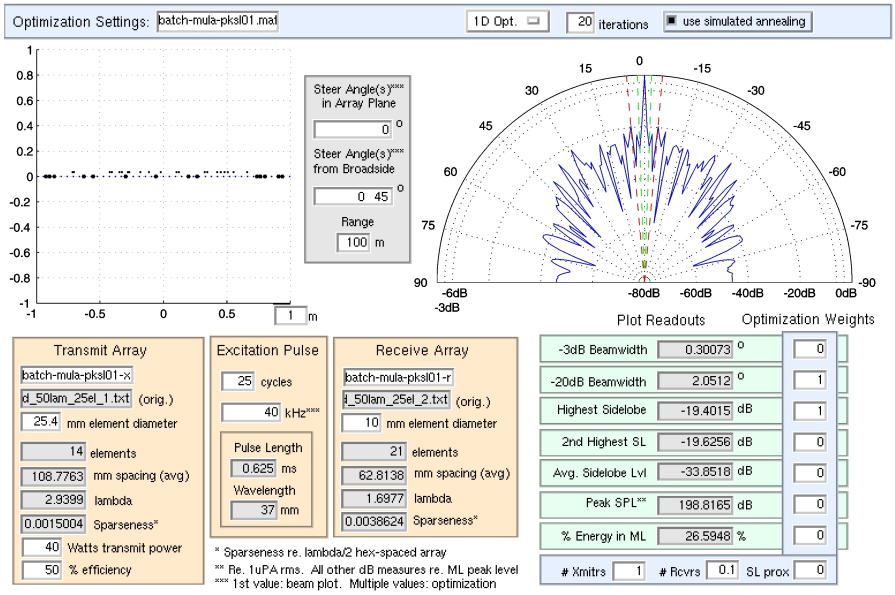


(a)

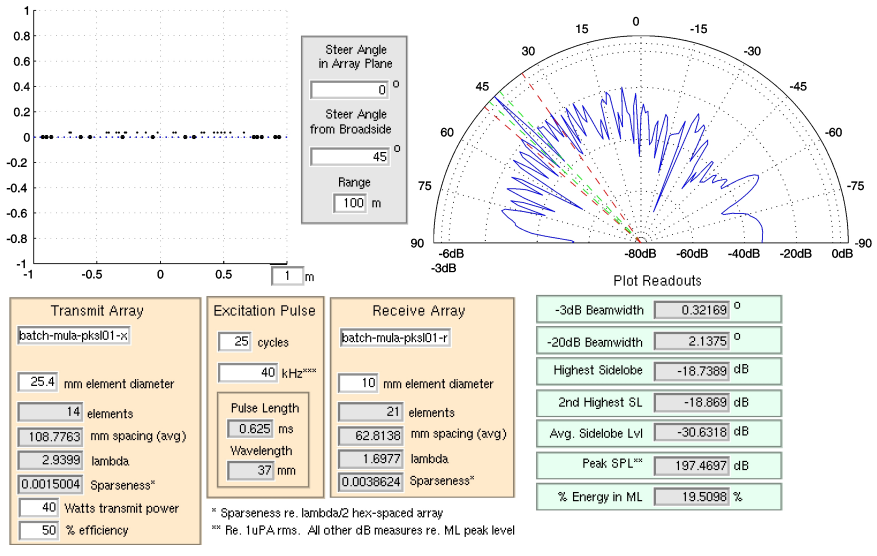


(b)

Figure 33: (a) 1D Array pair (optimized at broadside only) shows sidelobes 21dB lower than the main lobe at 0°. (b) Same array pair with main lobe steered to +45° shows a peak sidelobe 17dB lower than the main lobe (at +45°)



(a)



(b)

Figure 34: (a) Array pair (optimized at broadside and 45° from broadside) shows sidelobes 19dB lower than the main lobe at 0°. (b) Same array pair with main lobe steered to +45° shows a peak sidelobe also 19dB lower than the main lobe (at +45°)

5.2.2 Summary

Table 9 summarizes array pair performance when comparing optimization at one steer angle to optimization at two.

Steer Angle	0°			45°		
Figure Number	32	33	34	32	33	34
-3dB Beamwidth (°)	0.15	0.08	0.30	0.43	0.38	0.32
-20dB Beamwidth (°)	1.01	0.55	2.05	2.8	4.02	2.14
Highest SL (dB)	-3.4	-21.6	-19.4	2.4	-16.8	-18.7
Avg. SL (dB)	-31.9	-33.4	-33.9	-35.3	-31.8	-30.6
Transmit SPL (dB)	201.3	198.5	198.8	201.1	197.1	197.5
% Energy in ML	21.1	20.7	26.6	32.8	26.4	19.5
Transmit Elements	25	13	14	25	13	14
Receive Elements	25	29	21	25	29	21

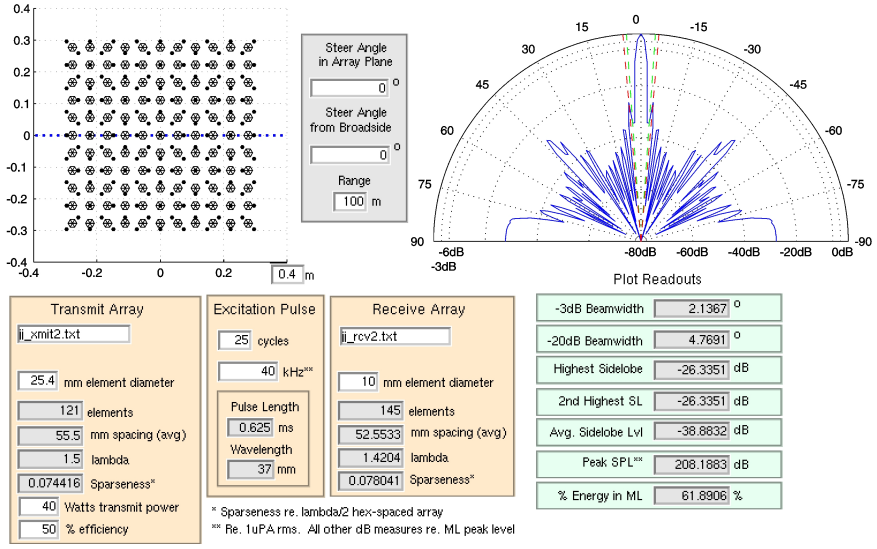
Table 9: Summary of multiple-angle optimization results

5.3 2D Optimizations

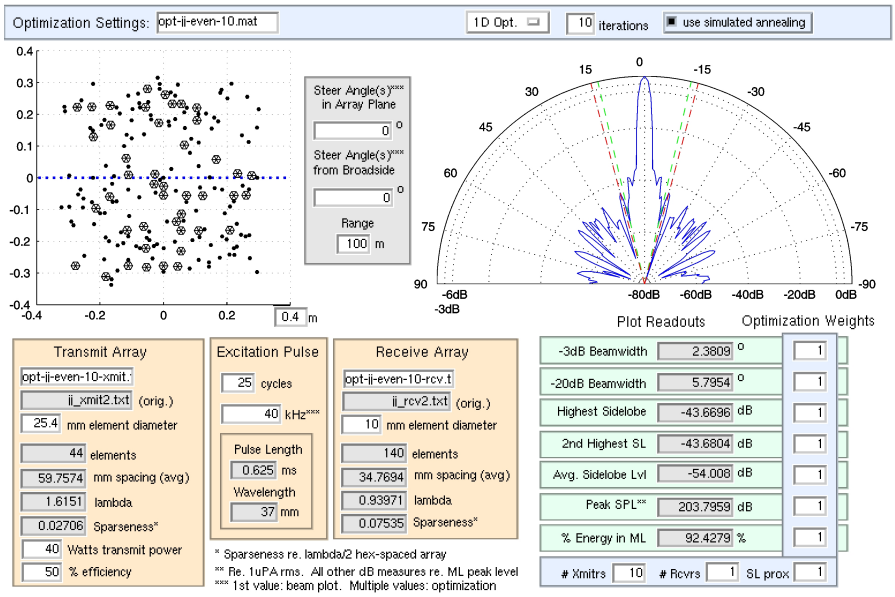
2D Simulations were run in Matlab 5.3 Linux on an Intel Pentium III 1.0GHz PC. Each optimization took approximately 3 hours to run to completion on the specified hardware.

5.3.1 Diagonally-Optimized Vernier Array

Starting with the diagonally optimized verier array presented in section 3.3.2, a simple attempt was made at further optimization using 10 iterations of simulated annealing. Individual cost functions were all weighted evenly, except for the transmitter element cost, which was given a weight 10 times as great as the others. The results are shown in Figure 35. While the -3dB and -20dB beamwidths increased by 0.2° and 1.0° respectively, the peak sidelobes were reduced by 17dB, the average sidelobe level was reduced by 17dB, the main lobe energy ratio increased by 31% to 92.4%, and the number of transmit elements was reduced from 121 to 44. These are substantial improvements on an "optimized" array in relatively few iterations.



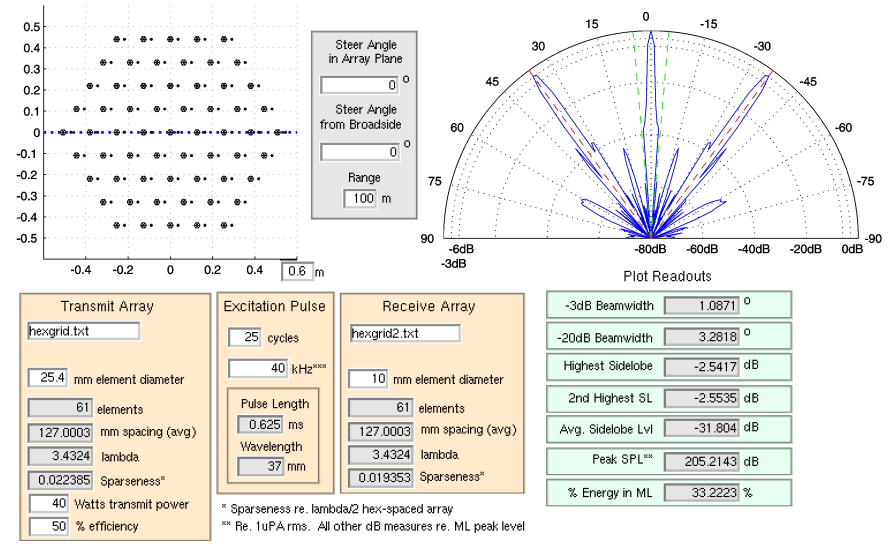
(a)



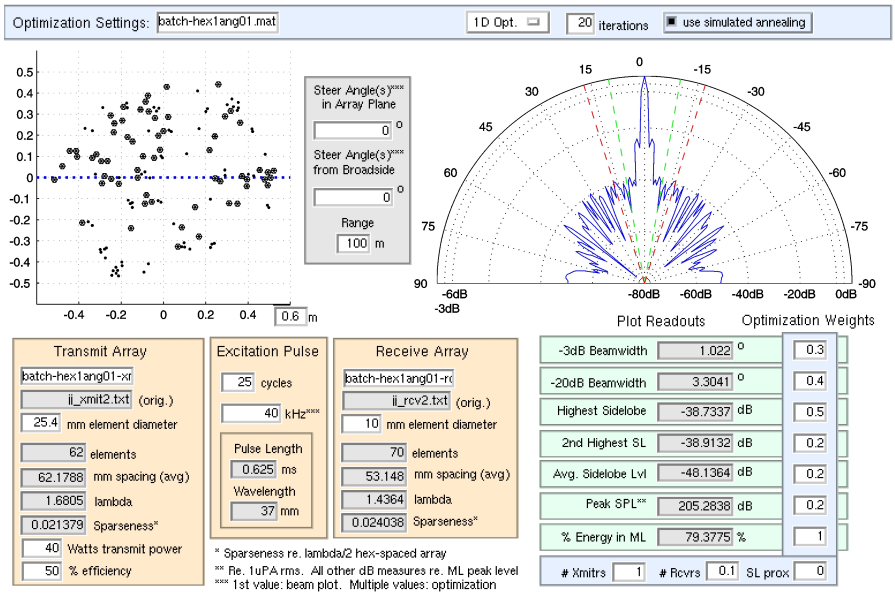
(b)

Figure 35: Diagonally-optimized Vernier array, (a) before and (b) after optimization with reduced sidelobes

5.3.2 Hex Array



(a)

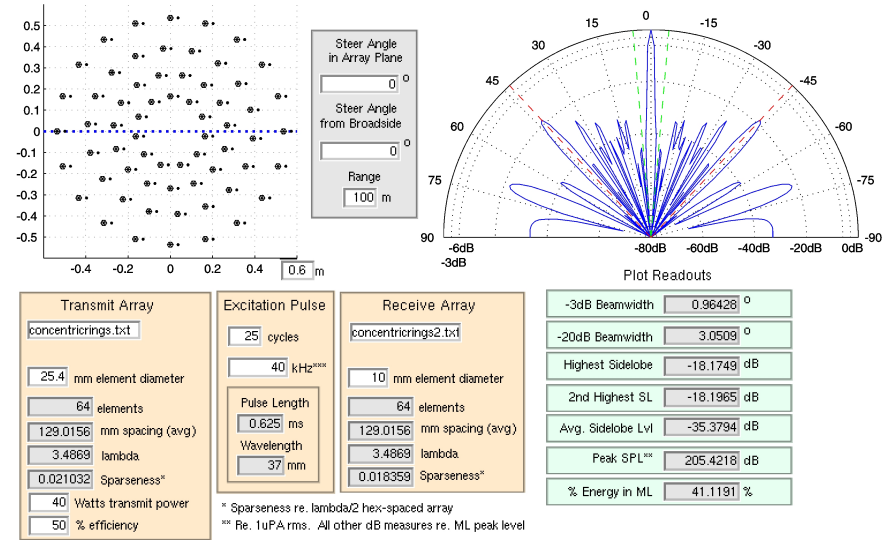


(b)

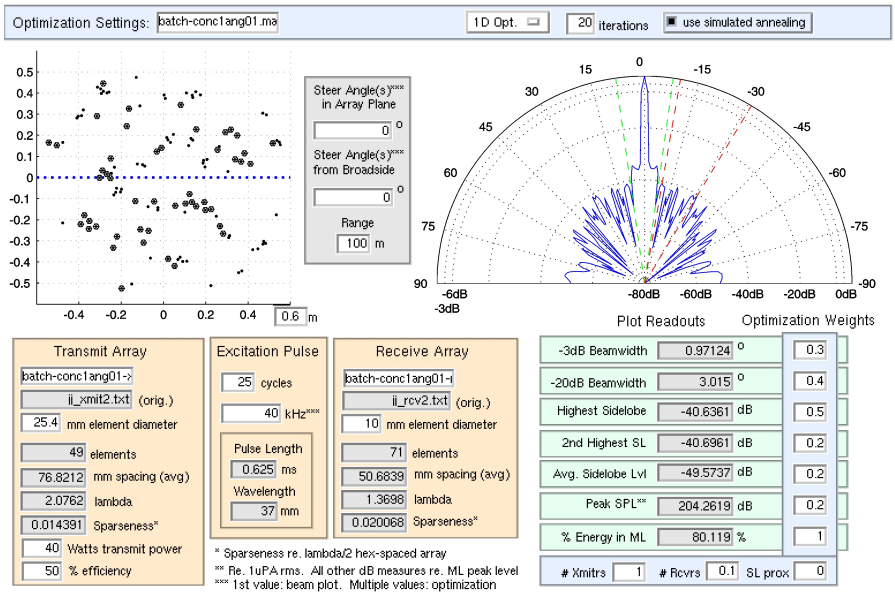
Figure 36: Hexagonal array, (a) before and (b) after optimization again with reduced sidelobes

The hexagon-aperture hexagonally-spaced array from Section 3.4.1 was used as another starting point for a 2D array pair optimization test. As with the 1D optimization examples, the same hexagonal array was used as the transmit and receive array, with the receiving array displaced slightly to avoid initial element overlap. In setting the weights for the cost functions, this time a non-uniform set of weights was used in an attempt to simulate a realistic scenario where certain array performance needs would dictate the method of carrying out the optimization. 20 iterations of simulated annealing were used, and the results are displayed in Figure 36. The -3dB and -20dB beamwidths remained virtually the same, as did the number of transmit elements, despite the high transmitter cost used in optimization. The vast improvements came in the form of greatly reduced sidelobe levels, (36dB lower) average sidelobe level (17dB reduction), and a greatly increased main lobe energy ratio (36% increase to 79%). Other than the high transmitter cost, which was likely counteracted by the SPL cost, the most highly weighted cost parameters were the main lobe energy ratio and the peak sidelobe level. As both of these parameters improved considerably without a great negative affect on the other parameters, the optimization was deemed successful.

5.3.3 Concentric Ring Array



(a)



(b)

Figure 37: Concentric ring array, (a) before and (b) after optimization showing reduced sidelobes and fewer transmit elements

As with the hexagonal array pair, the concentric-rings array from Section 3.4.2 was used as a transmit and receive array pair starting point for optimization. Identical cost function weights to the ones used for the hexagonal array were used to optimize the concentric ring array pair. Figure 37 shows the results from 20 iterations of simulated annealing. Again, virtually no changes were seen in the -3dB and -20dB beamwidth measures. However, unlike the constant element numbers of the hexagonal array pair optimization, the optimized concentric rings array pair ended with 15 fewer transmitters and 7 more receivers. This time, the three most heavily weighted cost parameters all saw significant improvements. The number of transmit elements was reduced 23%, the highest sidelobe level was reduced by 22dB, and the main lobe energy ratio climbed 39% to 80%. Another successful optimization.

6 Conclusions

Analysis of acoustic imaging fundamentals, as well as bistatic transmit and receive array beam patterns has revealed desirable beam pattern properties that can be achieved with dense arrays. As high element numbers in acoustic arrays are cumbersome with respect to manufacture and signal processing, techniques for reduction of array element numbers were reviewed. Several array patterns offer element reduction, but at a great cost in beam quality. Array optimization is shown to be both highly desirable in the improvement of beam properties and readily attainable through straightforward iterative simulated annealing methods. The highly flexible array cost measure implemented has allowed optimized array pairs to be generated based on a user-defined set of desired array properties. Optimized array pairs consistently saw substantial improvements in beam properties compared to the unoptimized cases, proving the validity and effectiveness of the optimization methods used.

6.1 System Limitations

6.1.1 Long Computation Times

The greatest obstacle to using the optimization algorithms presented in this thesis to a greater potential lies in the time-intensive calculations performed by Field II in

determining the emitted field by an array of arbitrary geometry. In Matlab 5.3 on Windows platforms, further substantial reductions in speed were witnessed, making the array optimization package difficult to port.

6.2 Further Development

6.2.1 3D Plotting

Currently for each new array pair considered, the resulting beam pattern is analyzed as a 1D cross-section of a 2D beam pattern. Ideally analysis of the beam patterns for 2D array pairs would have taken into consideration the entire hemispherical beam pattern rather than an arc at a specific angle in the array plane. 2D beam analysis was deemed to be too computationally intensive using the current methods of simulation, but could be implemented as a logical extension to the principles already in place.

6.2.2 Beamwidth Measurement Accuracy

By implementing a 2nd or 3rd order interpolation scheme for beamwidth calculations, these absolute values will be more accurate, especially in the case where there exist sharp discontinuities in the in the slope of the beam pattern. This is often the case near the main lobe of a tightly focused beam. Alternatively, beam plot measurements can be calculated on a finer scale close to the main lobe and more coarsely in the sidelobe regions. This would allow for finer beamwidth measurement without adding to greatly to the number of computations required to perform optimization.

6.2.3 Code Optimization

While Field II provides a reliable method for array pair simulations, it suffers from relatively slow computation times and platform dependence beyond that of Matlab. By adapting the mathematical principles of Huygen's principle used by Field II, perhaps the beam calculations could be carried out more rapidly. Furthermore, these calculations can be optimized to take advantage of parallel processing of beam profile data to further reduce computation times.

A Array Plotter Interface Code

A.1 User Interface Code

```
%%% Robert McPhie
%%% Sept 10, 2002
%%% ap_gui.m
%%% This is the callback function to handle all user interaction
%%% with the array plotter GUI.
function ap_gui(action)
global Th
global Th_xmit
global Th_rcv
global Th_orig_xmit
global Th_orig_rcv
global hp_xmit
global hp_rcv
global hhp
global beam_pat
global c
global data
global new_x_array
global new_r_array
global APFIG
global STATFIG
global MONFIG
switch(action)
     case 'initialize'
        APFIG=hgload('ap_gui_display.fig');
        axes(findobj(APFIG, 'Tag', 'ac_plot'));
        cla;

     case 'save_as'
        [fig_filename, fig_pathname] = uiputfile('*.fig','Save Figure As:');
        hgsave(APFIG, fullfile(fig_pathname, fig_filename));

    case 'ac_xmit_filebrowse'
        [ac_xmit_filename, ac_xmit_pathname] = uigetfile('*.txt','Transmit
            Array Configuration File');

        set(findobj(APFIG,'Tag','ac_xmit_filename_text'),'String',
            ac_xmit_filename);


    case 'ac_rcv_filebrowse'
```

```
    [ac_rcv_filename, ac_rcv_pathname] = uigetfile('*.txt','Receive Array
        Configuration File');

    set(findobj(APFIG,'Tag','ac_rcv_filename_text'), 'String',
        ac_rcv_filename);

case 'ac_xmit_edit'
    ac_xmit_filename = get(findobj(APFIG,'Tag', 'ac_xmit_filename_text'),
        'String');
    array_centerpoints=load(ac_xmit_filename);

    set(findobj(APFIG,'Tag','readout_xmit_elements'),'String',
        num2str(size(array_centerpoints,1)));
    spc=avgspace(array_centerpoints);

    set(findobj(APFIG,'Tag','readout_xmit_spacing_mm'), 'String',
        num2str(spc*1e3));

    wl=str2double(get(findobj(APFIG,'Tag', 'readout_wavelength'),
        'String'))*1e-3;
    freq=c/wl;
    r=max(sqrt(array_centerpoints(:,1).^2+array_centerpoints(:,2).^2));
    sparceness=size(array_centerpoints,1)/size(makehexgrid(freq, c, 0.5,
        r),1);

    set(findobj(APFIG,'Tag','readout_xmit_spacing_lambda'),'String',
        num2str(spc/wl));

    set(findobj(APFIG,'Tag','readout_xmit_sparceness'), 'String',
        num2str(sparceness));

    D=str2double(get(findobj(APFIG,'Tag','ac_xmit_diameter'),
        'String'))*1e-3;
    array_element_verteces=xdc_verteces_sub6(array_centerpoints,D);

    ac_range=str2double(get(findobj(APFIG,'Tag', 'ac_range_text'),
        'String'));

    ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
        'ac_steerangle_ap_text'), 'String'));
    ac_steerangle_ap=ac_steerangle_ap(1);

    ac_steerangle_bs=str2num(get(findobj(APFIG,'Tag',
        'ac_steerangle_bs_text'), 'String'));
    ac_steerangle_bs=ac_steerangle_bs(1);
    [fpx, fpy, fpz] = sph2cart(deg2rad(ac_steerangle_ap),
        pi/2+deg2rad(ac_steerangle_bs), ac_range);
    Th_xmit = xdc_triangles(array_element_verteces, array_centerpoints,
```

```
        [fpx fpy fpz]);

case 'ac_rcv_edit'
    ac_rcv_filename = get(findobj(APFIG, 'Tag', 'ac_rcv_filename_text'),
        'String');
    array_centerpoints=load(ac_rcv_filename);

    set(findobj(APFIG, 'Tag', 'readout_rcv_elements'), 'String',
        num2str(size(array_centerpoints, 1)));
    spc=avgspace(array_centerpoints);

    set(findobj(APFIG, 'Tag', 'readout_rcv_spacing_mm'), 'String',
        num2str(spc*1e3));

    wl=str2double(get(findobj(APFIG, 'Tag', 'readout_wavelength'),
        'String'))*1e-3;
    freq=c/wl;
    r=max(sqrt(array_centerpoints(:,1).^2+array_centerpoints(:,2).^2));
    sparceness=size(array_centerpoints,1)/size(makehexgrid(freq, c, 0.5,
        r),1);

    set(findobj(APFIG, 'Tag', 'readout_rcv_spacing_lambda'), 'String',
        num2str(spc/wl));

    set(findobj(APFIG, 'Tag', 'readout_rcv_sparceness'), 'String',
        num2str(sparceness));

    D=str2double(get(findobj(APFIG, 'Tag', 'ac_rcv_diameter'),
        'String'))*1e-3;
    array_element_verteces=xdc_verteces_sub6(array_centerpoints,D);

    ac_range=str2double(get(findobj(APFIG, 'Tag', 'ac_range_text'),
        'String'));

    ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
        'ac_steerangle_ap_text'), 'String'));
    ac_steerangle_ap=ac_steerangle_ap(1);

    ac_steerangle_bs=str2num(get(findobj(APFIG, 'Tag',
        'ac_steerangle_bs_text'), 'String'));
    ac_steerangle_bs=ac_steerangle_bs(1);
    [fpx, fpy, fpz] = sph2cart(deg2rad(ac_steerangle_ap),
        pi/2+deg2rad(ac_steerangle_bs), ac_range);
    Th_rcv = xdc_triangles(array_element_verteces, array_centerpoints,
        [fpx fpy fpz]);

case 'ac_plot_az_line'
```

```
    axes(findobj(APFIG,'Tag','ac_plot'));
    hold on
    if findobj(APFIG, 'Tag', 'az_line_tag')
       delete(findobj(APFIG, 'Tag', 'az_line_tag'));
    end

    ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
        'ac_steerangle_ap_text'), 'String'));
    ac_steerangle_ap=ac_steerangle_ap(1);

    az=deg2rad(ac_steerangle_ap);
    xlim=get(findobj(APFIG, 'Tag', 'ac_plot'), 'XLim');
    ylim=get(findobj(APFIG, 'Tag', 'ac_plot'), 'YLim');
    if az == 0
       pts_x=xlim;
       pts_y=[0 0];
    elseif abs(az-pi/2) < 0.01
       pts_x=[0 0];
       pts_y=ylim;
    elseif xlim(2)/cos(az) <= ylim(2)/sin(az)
       pts_x=xlim;
       pts_y=[xlim(1)*tan(az) xlim(2)*tan(az)];
    else
       pts_x=[ylim(1)/tan(az) ylim(2)/tan(az)];
       pts_y=ylim;
    end
    az_line_pointer=plot(pts_x, pts_y, 'b:');
    set(az_line_pointer,'Tag','az_line_tag');
    set(az_line_pointer, 'LineWidth', 2);
    set(gca, 'Tag', 'ac_plot');


case 'ac_plot'

    ac_plot_type = get(findobj(APFIG, 'Tag','ac_popup'),'Value');
    %ac_plot_se = get(findobj(APFIG, 'Tag', 'sub_elements_checkbox'),
        'Value');
    ac_plot_se = 1;
    ac_plot_orig_arrays = get(findobj(APFIG, 'Tag',
        'orig_arrays_checkbox'), 'Value');
    offset_checkbox = get(findobj(APFIG, 'Tag', 'offset_rcv_checkbox'),
        'Value');

    DX=str2double(get(findobj(APFIG, 'Tag', 'ac_xmit_diameter'),
        'String'))*1e-3;

    DR=str2double(get(findobj(APFIG, 'Tag', 'ac_rcv_diameter'),
        'String'))*1e-3;
```

```
   ac_plot_autoscale = get(findobj(APFIG, 'Tag', 'ac_plot_autoscale'),
       'Value');

   axes(findobj(APFIG,'Tag','ac_plot'));
   hold on

   if findobj(APFIG, 'Tag', 'xmit_plot_tag')
      delete(findobj(APFIG, 'Tag', 'xmit_plot_tag'));
   end
   if findobj(APFIG, 'Tag', 'rcv_plot_tag')
      delete(findobj(APFIG, 'Tag', 'rcv_plot_tag'));
   end
   if findobj(APFIG, 'Tag', 'orig_xmit_plot_tag')
      delete(findobj(APFIG, 'Tag', 'orig_xmit_plot_tag'));
   end
   if findobj(APFIG, 'Tag', 'orig_rcv_plot_tag')
      delete(findobj(APFIG, 'Tag', 'orig_rcv_plot_tag'));
   end

   if ac_plot_orig_arrays
      arraydisp2_opt(Th_xmit, Th_orig_xmit, DX, Th_rcv, Th_orig_rcv, DR,
          ac_plot_type, offset_checkbox*(DR+DX), ac_plot_se);
   else
      arraydisp2(Th_xmit, DX, Th_rcv, DR, ac_plot_type,
          offset_checkbox*(DR+DX), ac_plot_se);
   end

   if ac_plot_autoscale
      axis auto;
      axis equal;
      ax_lim = axis;
      ax_lim = round(ax_lim(2)*10)/10;
      set(findobj(APFIG,'Tag','ac_plot_scale'),'String',num2str(ax_lim));
   else
      ax_lim = str2double(get(findobj(APFIG, 'Tag', 'ac_plot_scale'),
           'String'));
      axis([-ax_lim ax_lim -ax_lim ax_lim]);
   end

   set(gca,'Tag','ac_plot');

case 'xp_edit'
   freq = str2double(get(findobj(APFIG, 'Tag', 'xp_freq_text'),
       'String'))*1e3;
   cycles = str2double(get(findobj(APFIG, 'Tag', 'xp_cycles_text'),
       'String'));

   fs = 64*freq;
```

```
        set_field('fs', fs);

        set(findobj(APFIG, 'Tag', 'readout_pulse_length'), 'String',
            num2str(cycles/freq*1e3));

        set(findobj(APFIG, 'Tag', 'readout_wavelength'), 'String',
            num2str(c/freq*1e3));

        Single_Transmitter = xdc_triangles(oneinch([0 0 0]), [0 0 0],
            [0 0 1]);
        xdc_excitation(Single_Transmitter, sin(2*pi*[0:freq/fs:1]));
        [Single_hp, Single_start_time] = calc_hp(Single_Transmitter, [0 0 1]);
        field_SPL = max(Single_hp)/sqrt(2);

        % form is single_element_SPL(f0, phi_s, D, W, efficiency)
        DX=str2double(get(findobj(APFIG, 'Tag', 'ac_xmit_diameter'),
            'String'))*1e-3;
        powr=str2double(get(findobj(APFIG, 'Tag', 'ac_xmit_power'),
            'String'));

        eff=str2double(get(findobj(APFIG, 'Tag', 'ac_xmit_efficiency'),
            'String'))/100;

        c_steerangle_bs=str2num(get(findobj(APFIG, 'Tag',
            'ac_steerangle_bs_text'), 'String'));
        angs=ac_steerangle_bs(1);

        actual_SPL_dB = single_element_SPL(freq, 0, DX, powr, eff);

        scale_factor = (10^(actual_SPL_dB/10))*1e-6 / field_SPL;
        times = [0:1/fs:cycles/freq];
        pulsesig = sin(2*pi*times*freq);
        xdc_excitation(Th_xmit, pulsesig)
        xdc_excitation(Th_rcv, pulsesig)
        xdc_impulse(Th_xmit, scale_factor)
        xdc_impulse(Th_rcv, 1/scale_factor)

%        axes(findobj('Tag','xp_plot'));
%        cla;
%        plot(times,pulsesig);
%        xlabel('time (s)')
%        ylabel('Amp.')
%        set(gca,'Tag','xp_plot');

case 'plot_draw'
        ap_gui ac_xmit_edit;
        ap_gui ac_rcv_edit;
        ap_gui xp_edit;
```

```
axes(findobj(APFIG,'Tag','bp_plot'));
p_res = str2double(get(findobj(APFIG, 'Tag', 'plot_res_text'),
    'String'));

ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
    'ac_steerangle_ap_text'), 'String'));
az=deg2rad(ac_steerangle_ap(1));
ac_range = str2double(get(findobj(APFIG, 'Tag', 'ac_range_text'),
    'String'));
plot_type = get(findobj(APFIG,'Tag','plot_popup'),'Value');
if plot_type == 1
   [hp_xmit, start_time_xmit] = calc_hp(Th_xmit,
        radialgridmake(az, az, 1, 0, pi, 180/p_res+1, ac_range));
   beam_pat_1=hp_xmit;
   beam_pat_2=hp_xmit;
   save('testxmit2.mat','hp_xmit')
elseif plot_type == 2
   [hp_rcv, start_time_rcv] = calc_hp(Th_rcv,
        radialgridmake(az,az,1, 0, pi, 180/p_res+1, ac_range));
   beam_pat_1=hp_rcv;
   beam_pat_2=hp_rcv;
else
   [hp_xmit, start_time_xmit] = calc_hp(Th_xmit,
        radialgridmake(az,az,1, 0, pi, 180/p_res+1, ac_range));
   [hp_rcv, start_time_rcv] = calc_hp(Th_rcv,
        radialgridmake(az,az,1, 0, pi, 180/p_res+1, ac_range));
   beam_pat_1=hp_xmit;
   beam_pat_2=hp_rcv;
end

xp_freq = str2double(get(findobj(APFIG, 'Tag', 'xp_freq_text'),
    'String'))*1e3;
fs = xp_freq*64;
[Imax1, Jmax1]=find(beam_pat_1>.99*max(max(beam_pat_1)));
Icenter1 = median(Imax1);
Istart1 = round(Icenter1-(1/4/xp_freq*fs));
Iend1 = round(Icenter1+(3/4/xp_freq*fs));
beam_pat_sub1=beam_pat_1(Istart1:Iend1,:);
rms_bps1=(mean(beam_pat_sub1.^2)).^.5;

[Imax2, Jmax2]=find(beam_pat_2>.99*max(max(beam_pat_2)));
Icenter2 = median(Imax2);
Istart2 = round(Icenter2-(1/4/xp_freq*fs));
Iend2 = round(Icenter2+(3/4/xp_freq*fs));
beam_pat_sub2=beam_pat_2(Istart2:Iend2,:);
rms_bps2=(mean(beam_pat_sub2.^2)).^.5;

polardbplot2(rms_bps1, rms_bps2, plot_type);
```

```
   set(gca,'Tag','bp_plot');

case 'opt_filebrowse'
   [opt_filename, opt_pathname] = uigetfile('*.mat','Optimization File');
   set(findobj(APFIG,'Tag','opt_filename'),'String',opt_filename);
   ap_gui opt_load;


case 'opt_load'
  figure(APFIG)
   opt_filename = get(findobj('Tag','opt_filename'),'String')
   load(opt_filename);

   set(findobj(APFIG, 'Tag', 'opt_iter'), 'String',
       num2str(length(temp_array)));
   set(findobj(APFIG,'Tag','opt_sa'),'Value', sa);

   set(findobj(APFIG,'Tag','weight_xmit_el'),'String', wgts(1));
   set(findobj(APFIG,'Tag','weight_rcv_el'),'String', wgts(2));
   set(findobj(APFIG,'Tag','weight_xmit_lvl'),'String', wgts(3));
   set(findobj(APFIG,'Tag','weight_-3db_bw'),'String', wgts(4));
   set(findobj(APFIG,'Tag','weight_-20db_bw'),'String', wgts(5));
   set(findobj(APFIG,'Tag','weight_sl1'),'String', wgts(6));
   set(findobj(APFIG,'Tag','weight_sl2'),'String', wgts(7));
   set(findobj(APFIG,'Tag','weight_avg_sl'),'String', wgts(8));
   set(findobj(APFIG,'Tag','weight_perc_ml'),'String', wgts(9));
   set(findobj(APFIG,'Tag','weight_sl_prox'),'String', wgts(10));

   set(findobj(APFIG,'Tag','ac_xmit_filename_text'),'String',
       xmit_filename);
   set(findobj(APFIG,'Tag','ac_xmit_diameter'),'String',
       num2str(d_x*1e3));
   set(findobj(APFIG,'Tag','ac_xmit_power'),'String', num2str(pwr));
   set(findobj(APFIG,'Tag','ac_xmit_efficiency'),'String', num2str(eff));
   set(findobj(APFIG,'Tag','ac_rcv_filename_text'),'String',
       rcv_filename);
   set(findobj(APFIG,'Tag','ac_rcv_diameter'),'String',
       num2str(d_r*1e3));

   set(findobj(APFIG,'Tag','xp_freq_text'),'String',
       num2str(freqs.*1e-3));
   set(findobj(APFIG,'Tag','xp_cycles_text'),'String', num2str(cycles));

   set(findobj(APFIG,'Tag','ac_steerangle_ap_text'),'String',
       num2str(angs_ap));
   set(findobj(APFIG,'Tag','ac_steerangle_bc_text'),'String',
       num2str(angs_bs));
```

```
%load original xmit array

set(findobj(APFIG, 'Tag', 'orig_xmit_filename'), 'String',
    orig_xmit_filename);

array_centerpoints=load(orig_xmit_filename);

D=str2double(get(findobj(APFIG, 'Tag' , 'ac_xmit_diameter'),
    'String'))*1e-3;
array_element_verteces=xdc_verteces_sub6(array_centerpoints,D);

ac_range=str2double(get(findobj(APFIG, 'Tag', 'ac_range_text'),
    'String'));

ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
    'ac_steerangle_ap_text'), 'String'));
ac_steerangle_ap=ac_steerangle_ap(1);

ac_steerangle_bs=str2num(get(findobj(APFIG, 'Tag',
    'ac_steerangle_bs_text'), 'String'));
ac_steerangle_bs=ac_steerangle_bs(1);
[fpx, fpy, fpz] = sph2cart(deg2rad(ac_steerangle_ap),
    pi/2+deg2rad(ac_steerangle_bs), ac_range);
Th_orig_xmit = xdc_triangles(array_element_verteces,
    array_centerpoints, [fpx fpy fpz]);

%load original rcv array

set(findobj(APFIG, 'Tag', 'orig_rcv_filename'), 'String',
    orig_rcv_filename);

array_centerpoints=load(orig_rcv_filename);

D=str2double(get(findobj(APFIG, 'Tag', 'ac_rcv_diameter'),
    'String'))*1e-3;
array_element_verteces=xdc_verteces_sub6(array_centerpoints,D);

ac_range=str2double(get(findobj(APFIG, 'Tag', 'ac_range_text'),
    'String'));

ac_steerangle_ap=str2num(get(findobj(APFIG, 'Tag',
    'ac_steerangle_ap_text'), 'String'));
ac_steerangle_ap=ac_steerangle_ap(1);

ac_steerangle_bs=str2num(get(findobj(APFIG, 'Tag',
    'ac_steerangle_bs_text'), 'String'));
ac_steerangle_bs=ac_steerangle_bs(1);
[fpx, fpy, fpz] = sph2cart(deg2rad(ac_steerangle_ap),
```

```
        pi/2+deg2rad(ac_steerangle_bs), ac_range);
    Th_orig_rcv = xdc_triangles(array_element_verteces,
        array_centerpoints, [fpx fpy fpz]);


case 'opt_run'

    orig_xmit_filename = get(findobj(APFIG, 'Tag',
        'ac_xmit_filename_text'), 'String');
    d_x = str2num(get(findobj(APFIG, 'Tag', 'ac_xmit_diameter'),
        'String'))*1e-3;
    orig_rcv_filename = get(findobj(APFIG, 'Tag',
        'ac_rcv_filename_text'), 'String');
    d_r = str2num(get(findobj(APFIG, 'Tag', 'ac_rcv_diameter'),
        'String'))*1e-3;
    freqs = str2num(get(findobj(APFIG, 'Tag', 'xp_freq_text'),
        'String')).*1e3;
    cycles = str2num(get(findobj(APFIG,'Tag','xp_cycles_text'),'String'));
    pwr = str2num(get(findobj(APFIG,'Tag','ac_xmit_power'),'String'));
    eff = str2num(get(findobj(APFIG, 'Tag', 'ac_xmit_efficiency'),
        'String'));
    iter = str2num(get(findobj(APFIG,'Tag','opt_iter'),'String'));
    sa = get(findobj(APFIG,'Tag','opt_sa'),'Value');
    oned = 2-get(findobj(APFIG,'Tag','opt_dim_popup'), 'Value');
    opt_filename = 'working_opt_file.mat';
    wgts(1) = str2num(get(findobj(APFIG, 'Tag', 'weight_xmit_el'),
        'String'));
    wgts(2) = str2num(get(findobj(APFIG, 'Tag', 'weight_rcv_el'),
        'String'));
    wgts(3) = str2num(get(findobj(APFIG, 'Tag', 'weight_xmit_lvl'),
        'String'));
    wgts(4) = str2num(get(findobj(APFIG, 'Tag', 'weight_-3db_bw'),
        'String'));
    wgts(5) = str2num(get(findobj(APFIG, 'Tag', 'weight_-20db_bw'),
        'String'));
    wgts(6) = str2num(get(findobj(APFIG,'Tag','weight_sl1'),'String'));
    wgts(7) = str2num(get(findobj(APFIG,'Tag','weight_sl2'),'String'));
    wgts(8) = str2num(get(findobj(APFIG,'Tag','weight_avg_sl'),'String'));
    wgts(9) = str2num(get(findobj(APFIG, 'Tag', 'weight_perc_ml'),
        'String'));
    wgts(10)= str2num(get(findobj(APFIG, 'Tag', 'weight_sl_prox'),
        'String'));

    angs_ap = str2num(get(findobj(APFIG, 'Tag', 'ac_steerangle_ap_text'),
        'String'));
    angs_bs = str2num(get(findobj(APFIG, 'Tag', 'ac_steerangle_bs_text'),
        'String'));
    range = str2num(get(findobj(APFIG,'Tag','ac_range_text'),'String'));
```

```
    [new_x_array, new_r_array] = opt_2dff_mul(orig_xmit_filename, d_x,
        orig_rcv_filename, d_r, ...
        c, freqs, cycles, pwr, eff, ...
        iter, sa, oned, opt_filename, wgts, ...
        angs_ap, angs_bs, range);

    set(findobj(APFIG,'Tag','opt_filename'), 'String',
        strcat(opt_filename(1:end-4), '-done.mat'));
    ap_gui opt_save;
    ap_gui opt_load;
    ap_gui ac_xmit_edit;
    ap_gui ac_rcv_edit;
    ap_gui ac_plot;
    ap_gui plot_draw;

case 'opt_savebrowse'
    [new_opt_filename, new_opt_path] = uiputfile('*.mat',
        'Save Optimization File As...');
    set(findobj(APFIG,'Tag','opt_filename'), 'String', new_opt_filename);
    ap_gui opt_save;

case 'opt_save'
    load('working_opt_file-done.mat');
    opt_filename=get(findobj(APFIG, 'Tag', 'opt_filename'),'String');
    xmit_filename=strcat(opt_filename(1:end-4),'-xmit.txt')
    rcv_filename=strcat(opt_filename(1:end-4),'-rcv.txt')

    save(opt_filename, 'xmit_array', 'd_x', 'xmit_data', 'rcv_array',
        'd_r', 'rcv_data', ...
        'c', 'freqs', 'cycles', 'pwr', 'eff', ...
        'cur_cost', 'temp_array', 'cur_temp', 'cur_iter', 'cur_ele',
        'x_nr', 'sa', 'oned', ...
        'wgts', 'r_aptr', 'angs_ap', 'angs_bs', ...
        'opt_filename', 'orig_xmit_filename', 'orig_rcv_filename',
        'xmit_filename', 'rcv_filename');

    save(xmit_filename,'xmit_array','-ascii');
    save(rcv_filename,'rcv_array','-ascii');

    set(findobj(APFIG,'Tag','ac_xmit_filename_text'), 'String',
        xmit_filename);
    set(findobj(APFIG,'Tag','ac_rcv_filename_text'), 'String',
        rcv_filename);

 case 'close_arrayplotter'
    field_end;
```

```
        close(APFIG)
        if ishandle(STATFIG)
           close(STATFIG)
        end
        if ishandle(MONFIG)
           close(MONFIG)
        end
end
```

A.2 Beam Plotting

```
function polardbplot2(orig_xmit_data, orig_rcv_data, ptype)
global data
global APFIG
if ptype == 1
   orig_data = orig_xmit_data;
elseif ptype == 2
   orig_data = orig_rcv_data;
else
   orig_data = orig_xmit_data.*orig_rcv_data;
end
%clear axis and prepare for some custom plotting
cla
axis off
axis auto
steps=length(orig_data);
ang_incr=pi/(steps-1);
ang_incr_deg=180/(steps-1);
angles=[0:ang_incr:pi];
ac_steerangle=str2num(get(findobj(APFIG,'Tag','ac_steerangle_bs_text'),
    'String'));
ac_steerangle=ac_steerangle(1);
steerangle_index=(ac_steerangle+90)/ang_incr_deg+1;
datamax=20*log10(max(orig_data)/min(orig_data));
lo=0.8; % small proportional value for Label Offsets
data=20*log10(orig_data/min(orig_data));
fdr=get(findobj(APFIG,'Tag','bp_auto_scale'),'Value');  % full dynamic range
                                                 disabled (restricted to -80dB)
if fdr == 0
   plot_data=data-datamax+80;
   for jj=1:length(plot_data)
      if plot_data(jj)<0
         plot_data(jj)=0;
      end
   end
   plot_datamax=80;
else
   plot_data=data;
   plot_datamax=datamax;
end
plot_mainlobe_max=max(plot_data(steerangle_index));
mainlobe_max=max(data(steerangle_index));
[X_datamax, Y_datamax]=pol2cart([0:pi/100:pi],plot_datamax);
patch(X_datamax,Y_datamax,'w');
hold on
[X_max_0dB, Y_max_0dB]=pol2cart(angles,plot_mainlobe_max);
[X_max_3dB, Y_max_3dB]=pol2cart(angles,plot_mainlobe_max-3);
```

```
[x_3db_label, y_3db_label]=pol2cart(pi,plot_mainlobe_max-3);
text(x_3db_label-5*lo, y_3db_label-12*lo,'-3dB');
[X_max_6dB, Y_max_6dB]=pol2cart(angles,plot_mainlobe_max-6);
[x_6db_label, y_6db_label]=pol2cart(pi,plot_mainlobe_max-6);
text(x_6db_label-5*lo, y_6db_label-5*lo,'-6dB');
plot(X_max_0dB, Y_max_0dB, 'k:',X_max_3dB, Y_max_3dB, 'k:',X_max_6dB,
    Y_max_6dB, 'k:');
for i=0:-20:-plot_mainlobe_max
   [X_grid, Y_grid]=pol2cart(angles,plot_mainlobe_max+i);
   [x_label, y_label]=pol2cart(0,plot_mainlobe_max+i);
   if plot_mainlobe_max < 90 | mod(round(-i/20),2) == 1
      text(x_label-8*lo, y_label-5*lo,strcat(num2str(i),'dB'));
   end
   plot(X_grid, Y_grid, 'k:');
end
for i=-90:15:90
   [X_grid, Y_grid]=pol2cart((i+90)/180*pi,plot_datamax);
   plot([0 X_grid],[0 Y_grid],'k:');
   text(1.07*X_grid-4*lo,1.07*Y_grid,num2str(i));
end
[X,Y]=pol2cart(angles,plot_data);
plot(X,Y);
%SPL Readout
set(findobj(APFIG,'Tag','readout_SPL'),'String',
num2str(10*log10(max(orig_xmit_data)/1e-6)));
%Beamwidth Readout
indexmax=steerangle_index;
while data(indexmax) > mainlobe_max-3 & indexmax < length(data)
   indexmax=indexmax+1;
end
indexmin=steerangle_index;
while data(indexmin)> mainlobe_max-3 & indexmin > 1
   indexmin=indexmin-1;
end
imax=indexmax-(mainlobe_max-3-data(indexmax))/(data(indexmax-1) ...
    -data(indexmax));
imin=indexmin+(mainlobe_max-3-data(indexmin))/(data(indexmin+1) ...
    -data(indexmin));
beamwidth=(imax-imin)*ang_incr/pi*180;
set(findobj(APFIG,'Tag','readout_3dbbeamwidth'),'String',num2str(beamwidth));
%----------------------------------------
indexmax=steerangle_index;
while data(indexmax) > mainlobe_max-20 & indexmax < length(data)
   indexmax=indexmax+1;
end
indexmin=steerangle_index;
while data(indexmin)> mainlobe_max-20 & indexmin > 1
   indexmin=indexmin-1;
```

```
end
imax=indexmax-(mainlobe_max-20-data(indexmax))/(data(indexmax-1) ...
    -data(indexmax));
imin=indexmin+(mainlobe_max-20-data(indexmin))/(data(indexmin+1) ...
    -data(indexmin));
beamwidth20=(imax-imin)*ang_incr/pi*180;
set(findobj(APFIG,'Tag','readout_20dbbeamwidth'),'String', ...
    num2str(beamwidth20));
% Sidelobes Readouts
    % Peak & Null Detection
[vals, indeces]= maxnmin(data, 0.05*datamax); % peak detect threshold is
                                                5% of data range
crit_pts=[i2deg(indeces, ang_incr_deg)' indeces' vals'-mainlobe_max];
crit_pts_find_ml=sortrows([abs(crit_pts(:,2)-steerangle_index)
    crit_pts(:,2)],1);
iml=find(crit_pts(:,2)==crit_pts_find_ml(1,2));
    % Set values to zero if there are no peaks found
%if size(crit_pts(1:iml,1),1) <= 2
%   first_slp_minus_db=0;
%   first_slp_minus_ang=0;
%else
%   first_slp_minus_db=crit_pts(iml-2,3);
%   first_slp_minus_ang=crit_pts(iml-2,1);
%   [X_grid, Y_grid]=pol2cart((first_slp_minus_ang+90)/180*pi,plot_datamax);
%   plot([0 X_grid],[0 Y_grid],'r--');
%end
%
%if size(crit_pts(iml:end,1),1) <= 2
%   first_slp_plus_db=0;
%   first_slp_plus_ang=0;
%else
%   first_slp_plus_db=crit_pts(iml+2,3);
%   first_slp_plus_ang=crit_pts(iml+2,1);
%   [X_grid, Y_grid]=pol2cart((first_slp_plus_ang+90)/180*pi,plot_datamax);
%   plot([0 X_grid],[0 Y_grid],'r--');
%end
if size(crit_pts(1:iml,1),1) <= 1
   first_sln_minus_db=0;
   first_sln_minus_ang=0;
else
   first_sln_minus_db=crit_pts(iml-1,3);
   first_sln_minus_ang=crit_pts(iml-1,1);
   [X_grid, Y_grid]=pol2cart(deg2rad(first_sln_minus_ang+90),plot_datamax);
   plot([0 X_grid],[0 Y_grid],'g--');
end
if size(crit_pts(iml:end,1),1) <= 1
   first_sln_plus_db=0;
   first_sln_plus_ang=0;
```

```
else
   first_sln_plus_db=crit_pts(iml+1,3);
   first_sln_plus_ang=crit_pts(iml+1,1);
   [X_grid, Y_grid]=pol2cart(deg2rad(first_sln_plus_ang+90),plot_datamax);
   plot([0 X_grid],[0 Y_grid],'g--');
end
sorted_crit_pts=sortrows(crit_pts,3);
if size(sorted_crit_pts,1) < 3
   highest_slp_db=0;
   highest_slp_ang=0;
   highest2_slp_db=0;
   highest2_slp_ang=0;
else
   if sorted_crit_pts(end,2)==crit_pts(iml,2)
      highest_slp_db=sorted_crit_pts(end-1,3);
      highest_slp_ang=sorted_crit_pts(end-1,1);
      highest2_slp_db=sorted_crit_pts(end-2,3);
      highest2_slp_ang=sorted_crit_pts(end-2,1);
   elseif sorted_crit_pts(end-1,2)==crit_pts(iml,2)
      highest_slp_db=sorted_crit_pts(end,3);
      highest_slp_ang=sorted_crit_pts(end,1);
      highest2_slp_db=sorted_crit_pts(end-2,3);
      highest2_slp_ang=sorted_crit_pts(end-2,1);
      disp('Sidelobe higher than main lobe (1)');
   else
      highest_slp_db=sorted_crit_pts(end,3);
      highest_slp_ang=sorted_crit_pts(end,1);
      highest2_slp_db=sorted_crit_pts(end-1,3);
      highest2_slp_ang=sorted_crit_pts(end-1,1);
      disp('Sidelobe higher than main lobe (2)');
   end
   [X_grid, Y_grid]=pol2cart((highest_slp_ang+90)/180*pi,plot_datamax);

   plot([0 X_grid],[0 Y_grid],'r--');

   [X_grid, Y_grid]=pol2cart((highest2_slp_ang+90)/180*pi,plot_datamax);
   plot([0 X_grid],[0 Y_grid],'r--');
end

%set(findobj(APFIG,'Tag','readout_1stpp_db'),'String',
    num2str(first_slp_plus_db));
%set(findobj(APFIG,'Tag','readout_1stpp_angle'),'String',
    num2str(first_slp_plus_ang));
%set(findobj(APFIG,'Tag','readout_1stpm_db'),'String',
    num2str(first_slp_minus_db));
%set(findobj(APFIG,'Tag','readout_1stpm_angle'),'String',
    num2str(first_slp_minus_ang));
set(findobj(APFIG,'Tag','readout_high_db'),'String',num2str(highest_slp_db));
```

```
%set(findobj(APFIG,'Tag','readout_high_angle'),'String',
    num2str(highest_slp_ang));
set(findobj(APFIG,'Tag','readout_high2_db'),'String',
    num2str(highest2_slp_db));
%set(findobj(APFIG,'Tag','readout_high2_angle'),'String',
    num2str(highest2_slp_ang));
%set(findobj(APFIG,'Tag','readout_1stnp_db'),'String',
    num2str(first_sln_plus_db));
%set(findobj(APFIG,'Tag','readout_1stnp_angle'),'String',
    num2str(first_sln_plus_ang));
%set(findobj(APFIG,'Tag','readout_1stnm_db'),'String',
    num2str(first_sln_minus_db));
%set(findobj(APFIG,'Tag','readout_1stnm_angle'),'String',
    num2str(first_sln_minus_ang));
    % Sum main lobe & sidelobes, find energy ratio, and compute average
      sidelobe level
if size(crit_pts(1:iml,1),1) <= 1 | size(crit_pts(iml:end,1),1) <=1
   main_lobe_sum=0;
   no_main_lobe_sum=0;
   avg_sidelobe=0;
else
   main_lobe_sum=sum(orig_data(crit_pts(iml-1,2):crit_pts(iml+1,2)));
   no_main_lobe_sum=sum(orig_data)-main_lobe_sum;
   avg_sidelobe=20*log10(mean([orig_data(1:crit_pts(iml-1,2))
       orig_data(crit_pts(iml+1,2):length(orig_data))]) /
       orig_data(steerangle_index));
end
set(findobj(APFIG,'Tag','readout_int_main_lobe'),'String',
    num2str(main_lobe_sum));
set(findobj(APFIG,'Tag','readout_int_sidelobes'),'String',
    num2str(no_main_lobe_sum));
set(findobj(APFIG,'Tag','readout_percent_mainlobe'),'String',
    num2str(main_lobe_sum/(no_main_lobe_sum+main_lobe_sum)*100));
set(findobj(APFIG,'Tag','readout_avg_sidelobe'),'String',
    num2str(avg_sidelobe));
```

B Optimization Code

B.1 Initialization and Main Loop

```
function [new_x_array, new_r_array] = opt_2dff_mul(orig_xmit_filename, d_x,

orig_rcv_filename, d_r, ...
   c, freqs, cycles, pwr, eff, ...
   iter, sa, oned, opt_filename, wgts, ...
   angs_ap, angs_bs, range)

global STATFIG
global MONFIG
STATFIG = hgload('opt_status.fig');
MONFIG = figure;
xmit_array=load(orig_xmit_filename);
rcv_array=load(orig_rcv_filename);
remaining_elements=[ones(size(xmit_array,1),1) xmit_array;
zeros(size(rcv_array,1),1) rcv_array];
iter_array=[0:iter-1];
[cur_costs, xmit_data, rcv_data]=costs_2df_mul(xmit_array, d_x, rcv_array,
    d_r, c, ...
    freqs, cycles, pwr, eff, [0 0 0], 0, angs_ap, angs_bs, range);
temp_array=exp(-iter_array)-exp(-(iter-1))
cur_temp=temp_array(1);
cur_iter=1;
cur_ele=1;
x_nr=1;
cur_cost=costs_combiner2(cur_costs,wgts);
r_aptr=max(sqrt([xmit_array(:,1); rcv_array(:,1)].^2+[xmit_array(:,2);
    rcv_array(:,2)].^2 ));
xmit_filename=strcat(opt_filename(1:end-4), '-xmit.txt');
rcv_filename=strcat(opt_filename(1:end-4), '-rcv.txt');
opt_filename_n=strcat(opt_filename(1:end-4), '000000.mat');
save(opt_filename_n, 'xmit_array', 'd_x', 'xmit_data', 'rcv_array', 'd_r',
    'rcv_data', ...
    'remaining_elements', 'c', 'freqs', 'cycles', 'pwr', 'eff', ...
    'cur_cost', 'temp_array', 'cur_temp', 'cur_iter', 'cur_ele', 'x_nr',
    'sa', 'oned', ...
    'wgts', 'r_aptr', 'angs_ap', 'angs_bs', 'range',...
    'opt_filename', 'orig_xmit_filename', 'orig_rcv_filename',
    'xmit_filename', 'rcv_filename');

set(findobj(STATFIG, 'Tag', 'stat_cur_iter'), 'String', '1');
set(findobj(STATFIG, 'Tag', 'stat_total_iter'), 'String', num2str(iter));
set(findobj(STATFIG, 'Tag', 'stat_cur_el_xmit'), 'String', '1');
set(findobj(STATFIG, 'Tag', 'stat_cur_el_rcv'), 'String', '1');
```

```
set(findobj(STATFIG, 'Tag', 'stat_total_el_xmit'), 'String',
    num2str(size(xmit_array,1)));
set(findobj(STATFIG, 'Tag', 'stat_total_el_rcv'), 'String',
    num2str(size(rcv_array,1)));
set(findobj(STATFIG, 'Tag', 'stat_cur_temp'), 'String', num2str(cur_temp));
set(findobj(STATFIG, 'Tag', 'stat_orig_cost'), 'String', num2str(cur_cost));
set(findobj(STATFIG, 'Tag', 'stat_cur_cost'), 'String', num2str(cur_cost));
set(findobj(STATFIG, 'Tag', 'stat-3db'),'String', num2str(cur_costs(4)));
set(findobj(STATFIG, 'Tag', 'stat-20db'),'String', num2str(cur_costs(5)));
set(findobj(STATFIG, 'Tag', 'stat_sl1'),'String', num2str(cur_costs(6)));
set(findobj(STATFIG, 'Tag', 'stat_sl2'),'String', num2str(cur_costs(7)));
set(findobj(STATFIG, 'Tag', 'stat_avgsl'),'String', num2str(cur_costs(8)));
set(findobj(STATFIG, 'Tag', 'stat_spl'),'String', num2str(cur_costs(3)));
set(findobj(STATFIG, 'Tag', 'stat_perc'),'String', num2str(cur_costs(9)));
set(findobj(STATFIG, 'Tag', 'stat_prox'),'String', num2str(cur_costs(10)));
set(findobj(STATFIG, 'Tag', 'stat-3db_orig'),'String', num2str(cur_costs(4)));
set(findobj(STATFIG, 'Tag', 'stat-20db_orig'),'String',
    num2str(cur_costs(5)));
set(findobj(STATFIG, 'Tag', 'stat_sl1_orig'),'String', num2str(cur_costs(6)));
set(findobj(STATFIG, 'Tag', 'stat_sl2_orig'),'String', num2str(cur_costs(7)));
set(findobj(STATFIG, 'Tag', 'stat_avgsl_orig'),'String',
    num2str(cur_costs(8)));
set(findobj(STATFIG, 'Tag', 'stat_spl_orig'),'String', num2str(cur_costs(3)));
set(findobj(STATFIG, 'Tag', 'stat_perc_orig'),'String',
    num2str(cur_costs(9)));
set(findobj(STATFIG, 'Tag', 'stat_prox_orig'),'String',
    num2str(cur_costs(10)));

set(findobj(STATFIG, 'Tag', 'stat_cons_splitx'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_splitx'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_cons_movex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_movex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_cons_diex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_diex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_cons_splitx'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_splitx'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_cons_movex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_movex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_cons_diex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_acc_diex'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_high_accept'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_high_reject'), 'String', '0');
set(findobj(STATFIG, 'Tag', 'stat_low_accept'), 'String', '0');

done=0;
while done == 0
  [opt_filename_n, done]=opt_2dfp_mul(opt_filename_n, done);
end
```

```
load(opt_filename_n)
disp(opt_filename_n)
save(strcat(opt_filename_n(1:end-10),'-done.mat'), 'xmit_array', 'd_x',
    'xmit_data', 'rcv_array', 'd_r', 'rcv_data', ...
    'remaining_elements', 'c', 'freqs', 'cycles', 'pwr', 'eff', ...
    'cur_cost', 'temp_array', 'cur_temp', 'cur_iter', 'cur_ele', 'x_nr',
    'sa', 'oned', ...
    'wgts', 'r_aptr', 'angs_ap', 'angs_bs', 'range', ...
    'opt_filename', 'orig_xmit_filename', 'orig_rcv_filename',
    'xmit_filename', 'rcv_filename');

save(strcat(opt_filename,'-xmit.txt'), 'xmit_array', '-ascii');
save(strcat(opt_filename,'-rcv.txt'), 'rcv_array', '-ascii');
new_x_array=xmit_array;
new_r_array=rcv_array;
```

B.2 Simulated Annealing

```
function [outfilename, done] = opt_2dfp_mul(infilename, done)
global STATFIG
global MONFIG
% infile variables: xmit_array, d_x, xmit_data, rcv_array, d_r, rcv_data,
%                    remaining_elements, c, freqs, cycles, pwr, eff,
%                    cur_cost, temp_array, cur_temp, cur_iter, cur_ele, x_nr,
%                    sa, oned
%                    wgts, r_aptr, done, angs_ap, angs_bs, range
%                    opt_filename, orig_xmit_filename, orig_rcv_filename,
%                    xmit_filename, rcv_filename
%
%
load(infilename);
lambda=c/max(freqs);
lo2=lambda/2;
lo4=lambda/4;
% These probabilities are not followed exactly once the element overlap
% constraint is imposed
p_split=1/3;
p_death=1/3;
p_death=p_death/(1-p_split); % accounts for the fact that this is done
                             %only if p_split fails
new_el_valid=0;
cur_ele_full_n = ceil(rand*size(remaining_elements,1));
cur_ele_full = remaining_elements(cur_ele_full_n, :);
x_nr = cur_ele_full(1);
cur_ele = cur_ele_full(2:4);
set(findobj(STATFIG, 'Tag', 'stat_xmitr'), 'Value', x_nr);
set(findobj(STATFIG, 'Tag', 'stat_rcvr'), 'Value', 1-x_nr);
if x_nr
   array=xmit_array;
   d_array=d_x;
   other_array=rcv_array;
   d_other=d_r;
   dstr='Transmit ';
   d_cur=d_x;
   set(findobj(STATFIG, 'Tag', 'stat_cur_el_xmit'), 'String', ...
      num2str(size(xmit_array,1)-sum(remaining_elements(:,1))+1));
else
   array=rcv_array;
   d_array=d_r;
   other_array=xmit_array;
   d_other=d_x;
   dstr='Receive ';
   d_cur=d_r;
   set(findobj(STATFIG, 'Tag', 'stat_cur_el_rcv'), 'String', ...
```

```
num2str(size(rcv_array, 1) - size(remaining_elements, 1) +
    sum(remaining_elements(:, 1)) +1));

end
%ele_tot=num2str(size(array,1));
set(findobj(STATFIG, 'Tag', 'stat_cur_iter'), 'String', num2str(cur_iter));
set(findobj(STATFIG, 'Tag', 'stat_cur_temp'), 'String', num2str(cur_temp));
%disp(strcat('Iteration number:', num2str(cur_iter), ' of:', iter_tot));
%disp(strcat(dstr, ' element number:',num2str(cur_ele),' of:', ele_tot));
%disp(strcat('Temp:',num2str(cur_temp)));
new_array=array;
if rand < p_split
   %disp('Element splits?');

   if x_nr
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 1);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 0);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_splitx'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_splitx'),'String', num2str(n+1));
      acctag='stat_acc_splitx';
   else
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 1);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 0);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_splitr'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_splitr'),'String', num2str(n+1));
      acctag='stat_acc_splitr';
   end
   for i=1:10
      jmp_r=rand*lo2+d_cur;
      if oned==0
         jmp_theta=2*pi*rand;
      else
         if rand > 0.5
            jmp_theta=0;
         else
            jmp_theta=pi;
         end
      end
      [jmpx, jmpy] = pol2cart(jmp_theta, jmp_r);
```

```
      newx = cur_ele(1)+jmpx;
      newy = cur_ele(2)+jmpy;
      if sqrt(newx^2+newy^2) > r_aptr
         newx=cos(atan(newy/newx))*r_aptr;
         newy=sin(atan(newy/newx))*r_aptr;
      end

      if ~overlapping([newx newy 0], d_cur, array, d_array, other_array,
            d_other)
         %disp('non-overlapping split found');
         new_el_valid=1;
         new_array = [array; newx newy 0];
         break
      end
   end
end
if rand > p_death & new_el_valid == 0
   %disp('Element moves?');
   if x_nr
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 1);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 0);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_movex'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_movex'),'String', num2str(n+1));
      acctag='stat_acc_movex';
   else
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 1);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_movex'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_movex'),'String', num2str(n+1));
      acctag='stat_acc_movex';
   end
   for i=1:10
      jmp_r=rand*lo2;
      if oned==0
         jmp_theta=2*pi*rand;
      else
         if rand > 0.5
            jmp_theta=0;
         else
            jmp_theta=pi;
```

```
         end
      end
      [jmpx, jmpy] = pol2cart(jmp_theta, jmp_r);
      newx = cur_ele(1)+jmpx;
      newy = cur_ele(2)+jmpy;
      if sqrt(newx^2+newy^2) > r_aptr
         newx=cos(atan(newy/newx))*r_aptr;
         newy=sin(atan(newy/newx))*r_aptr;
      end

      no_cur_el_array = remove(cur_ele, array);

      if ~overlapping([newx newy 0], d_cur, no_cur_el_array, d_array,
            other_array, d_other)
         %disp('non-overlapping move found');
         new_el_valid=1;
         new_array=[no_cur_el_array; newx newy 0];
         break
      end
   end

end
if new_el_valid == 0
   %disp('Element dies?');
   if x_nr
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 1);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 0);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_diex'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_diex'),'String', num2str(n+1));
      acctag='stat_acc_diex';
   else
      set(findobj(STATFIG, 'Tag', 'stat_splitx'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_movex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_diex'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_splitr'), 'Value', 1);
      set(findobj(STATFIG, 'Tag', 'stat_mover'), 'Value', 0);
      set(findobj(STATFIG, 'Tag', 'stat_dier'), 'Value', 0);
      n=str2num(get(findobj(STATFIG, 'Tag', 'stat_cons_dier'),'String'));
      set(findobj(STATFIG, 'Tag', 'stat_cons_dier'),'String', num2str(n+1));
      acctag='stat_acc_dier';
   end
   if size(array,1) == 1
      new_array = array;
   else
```

```
      new_array = remove(cur_ele, array);
   end
end
figure(MONFIG)
subplot(234)
scatter(xmit_array(:,1), xmit_array(:,2))
axis equal;
subplot(235)
scatter(rcv_array(:,1), rcv_array(:,2))
axis equal;
remaining_elements = remove(cur_ele_full, remaining_elements);
if x_nr
   [cst, xmit_data, rcv_data] = costs_2df_mul(new_array, d_x, rcv_array, d_r,
      c, freqs, cycles, pwr, eff, rcv_array, rcv_data, angs_ap, angs_bs,
      range);
else
   [cst, xmit_data, rcv_data] = costs_2df_mul(xmit_array, d_x, new_array, d_r,
      c, freqs, cycles, pwr, eff, xmit_array, xmit_data, angs_ap, angs_bs,
      range);
end
new_cost=costs_combiner2(cst,wgts);
%disp(strcat('Old Cost:',num2str(cur_cost),'  New Cost:',num2str(new_cost)));
set(findobj(STATFIG, 'Tag', 'stat-3db'),'String', num2str(cst(4)));
set(findobj(STATFIG, 'Tag', 'stat-20db'),'String', num2str(cst(5)));
set(findobj(STATFIG, 'Tag', 'stat_sl1'),'String', num2str(cst(6)));
set(findobj(STATFIG, 'Tag', 'stat_sl2'),'String', num2str(cst(7)));
set(findobj(STATFIG, 'Tag', 'stat_avgsl'),'String', num2str(cst(8)));
set(findobj(STATFIG, 'Tag', 'stat_spl'),'String', num2str(cst(3)));
set(findobj(STATFIG, 'Tag', 'stat_perc'),'String', num2str(cst(9)));
set(findobj(STATFIG, 'Tag', 'stat_prox'),'String', num2str(cst(10)));
set(findobj(STATFIG, 'Tag', 'stat_new_cost'),'String', num2str(new_cost));
if new_cost < cur_cost
  kept_array=new_array;
  cur_cost=new_cost;
  %disp('array change to lower cost accepted')
  set(findobj(STATFIG, 'Tag', 'stat_cur_cost'),'String', num2str(new_cost));
  set(findobj(STATFIG, 'Tag', 'stat_lower'),'Value', 1);
  set(findobj(STATFIG, 'Tag', 'stat_higher'),'Value', 0);
  n=str2num(get(findobj(STATFIG, 'Tag', acctag),'String'));
  set(findobj(STATFIG, 'Tag', acctag),'String', num2str(n+1));
  n=str2num(get(findobj(STATFIG, 'Tag', 'stat_low_accept'),'String'));
  set(findobj(STATFIG, 'Tag', 'stat_low_accept'),'String', num2str(n+1));

elseif rand*cur_temp/10 > (new_cost-cur_cost)/cur_cost & sa
  kept_array=new_array;
  cur_cost=new_cost;
  %disp('array change to higher cost accepted')
  set(findobj(STATFIG, 'Tag', 'stat_cur_cost'),'String', num2str(new_cost));
```

```
  set(findobj(STATFIG, 'Tag', 'stat_lower'),'Value', 0);
  set(findobj(STATFIG, 'Tag', 'stat_higher'),'Value', 1);
  n=str2num(get(findobj(STATFIG, 'Tag', acctag),'String'));
  set(findobj(STATFIG, 'Tag', acctag),'String', num2str(n+1));
  n=str2num(get(findobj(STATFIG, 'Tag', 'stat_high_accept'),'String'));
  set(findobj(STATFIG, 'Tag', 'stat_high_accept'),'String', num2str(n+1));

else
  kept_array=array;
  %disp('array change to higher cost rejected')
  set(findobj(STATFIG, 'Tag', 'stat_lower'),'Value', 0);
  set(findobj(STATFIG, 'Tag', 'stat_higher'),'Value', 1);
  n=str2num(get(findobj(STATFIG, 'Tag', 'stat_high_reject'),'String'));
  set(findobj(STATFIG, 'Tag', 'stat_high_reject'),'String', num2str(n+1));
end
if x_nr
   xmit_array=kept_array;
   set(findobj(STATFIG, 'Tag', 'stat_total_el_xmit'), 'String',
       num2str(size(xmit_array,1)));
else
   rcv_array=kept_array;
   set(findobj(STATFIG, 'Tag', 'stat_total_el_rcv'), 'String',
       num2str(size(rcv_array,1)));
end
if isempty(remaining_elements)
   if cur_iter == length(temp_array)
      done = 1;
   else
      remaining_elements=[ones(size(xmit_array,1),1) xmit_array;
           zeros(size(rcv_array,1),1) rcv_array];
      cur_iter=cur_iter+1;
      cur_temp=temp_array(cur_iter);
   end
end
infilenum=str2double(infilename(end-9:end-4));
zpad=char(ones(1,6-length(num2str(infilenum+1))).*48);
outfilename=strcat(infilename(1:end-10),zpad,num2str(infilenum+1),'.mat');
save(outfilename, 'xmit_array', 'd_x', 'xmit_data', 'rcv_array', 'd_r',
    'rcv_data', ...
    'remaining_elements', 'c', 'freqs', 'cycles', 'pwr', 'eff', ...
    'cur_cost', 'temp_array', 'cur_temp', 'cur_iter', 'cur_ele', 'x_nr',
    'sa', 'oned', ...
    'wgts', 'r_aptr', 'angs_ap', 'angs_bs', 'range', ...
    'opt_filename', 'orig_xmit_filename', 'orig_rcv_filename',
    'xmit_filename', 'rcv_filename');
delete(infilename)
```

B.3 Cost Calculation

```
function [costs, xmit_data, rcv_data] = costs_2df_mul(xmit_array, d_xmit, ...
   rcv_array, d_rcv, c, pulse_freqs, pulse_cycles, power_xmit, eff_xmit, ...
   old_array, old_data, angs_ap, angs_bs, range)
global STATFIG
global MONFIG
data_bank=[];
xmit_data=[];
rcv_data=[];
ang_ap_vect=[];
ang_bs_vect=[];
pulse_freq_vect=[];
for i=1:length(angs_bs)
   for j=1:length(angs_ap)
      for k=1:length(pulse_freqs)
         ang_ap_vect=[ang_ap_vect angs_ap(j)];
         ang_bs_vect=[ang_bs_vect angs_bs(i)];
         pulse_freq_vect=[pulse_freq_vect pulse_freqs(k)];
      end
   end
end
oldisxmit=0;
oldisrcv=0;
if size(old_array) == size(xmit_array)
   if sortrows(old_array,[1 2]) == sortrows(xmit_array,[1 2])
      oldisxmit=1;
   end
end
if size(old_array) == size(rcv_array)
   if sortrows(old_array,[1 2]) == sortrows(rcv_array,[1 2])
      oldisrcv=1;
   end
end

for i=1:length(ang_ap_vect)
    ac_steerangle_ap=ang_ap_vect(i);
    ac_steerangle_bs=ang_bs_vect(i);
    ac_range=range;
    pulse_freq=pulse_freq_vect(i);
    %disp(strcat('AP Angle:',num2str(ac_steerangle_ap),'  BS
         Angle:',num2str(ac_steerangle_bs)));
    set(findobj(STATFIG, 'Tag', 'stat_ang_ap'), 'String',
         num2str(ac_steerangle_ap));
    set(findobj(STATFIG, 'Tag', 'stat_ang_bs'), 'String',
         num2str(ac_steerangle_bs));
    fs = 64*pulse_freq;
    set_field('fs', fs);
```

```
xmit_array_n=size(xmit_array,1);
rcv_array_n=size(rcv_array,1);

[fpx, fpy, fpz] = sph2cart(deg2rad(ac_steerangle_ap),
     pi/2+deg2rad(ac_steerangle_bs), ac_range);

xmit_element_verteces=xdc_verteces_sub6(xmit_array,d_xmit);
Th_xmit = xdc_triangles(xmit_element_verteces, xmit_array, [fpx fpy fpz]);
rcv_element_verteces=xdc_verteces_sub6(rcv_array,d_rcv);
Th_rcv = xdc_triangles(rcv_element_verteces, rcv_array, [fpx fpy fpz]);

%---------------------
[norx, nory, norz]= sph2cart(deg2rad(ac_steerangle_ap),
     pi/2+deg2rad(ac_steerangle_bs), 1);
Single_Transmitter = xdc_triangles(xdc_verteces_sub6([0 0 0],0.0254),
     [0 0 0], [norx nory norz]);
xdc_excitation(Single_Transmitter, sin(2*pi*[0:pulse_freq/fs:1]));
[Single_hp, Single_start_time] = calc_hp(Single_Transmitter, [0 0 1]);
field_SPL = max(Single_hp)/sqrt(2);

% form is single_element_SPL(f0, phi_s, D, W, efficiency)
actual_SPL_dB = single_element_SPL(pulse_freq, ac_steerangle_bs, d_xmit,
     power_xmit, eff_xmit);

scale_factor = (10^(actual_SPL_dB/10))*1e-6 / field_SPL;
times = [0:1/fs:pulse_cycles/pulse_freq];
pulsesig = sin(2*pi*times*pulse_freq);
xdc_excitation(Th_xmit, pulsesig)
xdc_excitation(Th_rcv, pulsesig)
xdc_impulse(Th_xmit, scale_factor)
xdc_impulse(Th_rcv, scale_factor)

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

p_res = 1;
az=deg2rad(ac_steerangle_ap);
%------------------------- TRANSMIT
%disp('XMIT');

if oldisxmit
   orig_data_xmit=old_data(i,:);
else
   [hp_xmit, start_time_xmit] = calc_hp(Th_xmit, radialgridmake(az,az, 1,
         0, pi, 180/p_res+1, ac_range));

   [Imax, Jmax]=find(hp_xmit>.99*max(max(hp_xmit)));
   Icenter = median(Imax);
   Istart = round(Icenter-(1/4/pulse_freq*fs));
```

```
   Iend = round(Icenter+(3/4/pulse_freq*fs));
   hp_xmit_sub=hp_xmit(Istart:Iend,:);
   orig_data_xmit=(mean(hp_xmit_sub.^2)).^0.5;

   figure(MONFIG)
   subplot(231)
   semilogy(orig_data_xmit)
end

peak_xmit_SPL=10*log10(max(orig_data_xmit)/1e-6);
peak_data=costs_peaks(orig_data_xmit, ac_steerangle_bs);
xhp_ang=peak_data(10);
xhp2_ang=peak_data(12);
%------------------------------------ RECEIVE
%disp('RCV');

if oldisrcv
   orig_data_rcv=old_data(i,:);
else
   [hp_rcv, start_time_rcv] = calc_hp(Th_rcv, radialgridmake(az,az, 1, 0,
        pi, 180/p_res+1, ac_range));

   [Imax, Jmax]=find(hp_rcv>.99*max(max(hp_rcv)));
   Icenter = median(Imax);
   Istart = round(Icenter-(1/4/pulse_freq*fs));
   Iend = round(Icenter+(3/4/pulse_freq*fs));
   hp_rcv_sub=hp_rcv(Istart:Iend,:);
   orig_data_rcv=(mean(hp_rcv_sub.^2)).^0.5;
   figure(MONFIG)
   subplot(232)
   semilogy(orig_data_rcv)
end

peak_data=costs_peaks(orig_data_rcv, ac_steerangle_bs);
rhp_ang=peak_data(10);
rhp2_ang=peak_data(12);
%------------------------------------ ANGULAR PROXIMITY OF PEAK SIDELOBES
slp = min(abs([xhp_ang-rhp_ang xhp_ang-rhp2_ang xhp2_ang-rhp_ang
     xhp2_ang-rhp2_ang]));
%------------------------------------ PULSE ECHO

%disp('PULSE-ECHO');

orig_data_pe=orig_data_xmit.*orig_data_rcv;

figure(MONFIG)
subplot(233)
semilogy(orig_data_pe)
```

```
    peak_data=costs_peaks(orig_data_pe, ac_steerangle_bs);
    first_slp_plus_db=peak_data(1);
    first_slp_plus_ang=peak_data(2);
    first_slp_minus_db =peak_data(3);
    first_slp_minus_ang=peak_data(4);
    first_sln_plus_db =peak_data(5);
    first_sln_plus_ang=peak_data(6);
    first_sln_minus_db =peak_data(7);
    first_sln_minus_ang=peak_data(8);
    highest_slp_db =peak_data(9);
    highest_slp_ang=peak_data(10);
    highest2_slp_db =peak_data(11);
    highest2_slp_ang=peak_data(12);
    avg_sidelobe = peak_data(13);
    percent_mainlobe = peak_data(14);

    steps=length(orig_data_pe);
    ang_incr=pi/(steps-1);
    ang_incr_deg=180/(steps-1);

    angles=[0:ang_incr:pi];

    steerangle_index=(ac_steerangle_bs+90)/ang_incr_deg+1;

    datamax=20*log10(max(orig_data_pe)/min(orig_data_pe));
    data_pe=20*log10(orig_data_pe/min(orig_data_pe));
    mainlobe_max=max(data_pe(steerangle_index));

    %Beamwidth Readout

    indexmax=steerangle_index;
    while data_pe(indexmax) > mainlobe_max-3 & indexmax < length(data_pe)
       indexmax=indexmax+1;
    end

    indexmin=steerangle_index;
    while data_pe(indexmin)> mainlobe_max-3 & indexmin > 1
       indexmin=indexmin-1;
    end


imax=indexmax-(mainlobe_max-3-data_pe(indexmax))/(data_pe(indexmax-1) -
     data_pe(indexmax));


imin=indexmin+(mainlobe_max-3-data_pe(indexmin))/(data_pe(indexmin+1) -
     data_pe(indexmin));
```

```
    beamwidth=(imax-imin)*ang_incr/pi*180;

    %----------------------------------------
    indexmax=steerangle_index;
    while data_pe(indexmax) > mainlobe_max-20 & indexmax < length(data_pe)
       indexmax=indexmax+1;
    end

    indexmin=steerangle_index;
    while data_pe(indexmin)> mainlobe_max-20 & indexmin > 1
       indexmin=indexmin-1;
    end


imax=indexmax - (mainlobe_max -20 -data_pe(indexmax))/(data_pe(indexmax - 1)
       -data_pe(indexmax));

imin=indexmin +(mainlobe_max -20 -data_pe(indexmin))/(data_pe(indexmin +1)
       -data_pe(indexmin));
    beamwidth20=(imax-imin)*ang_incr/pi*180;
    %----------------------------------------

    xmit_data=[xmit_data; orig_data_xmit];
    rcv_data=[rcv_data; orig_data_rcv];

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

    data_bank = [data_bank; ...
            xmit_array_n, ...           %  1
            rcv_array_n, ...            %  2
            peak_xmit_SPL, ...          %  3
            beamwidth, ...              %  4
            beamwidth20, ...            %  5
            first_slp_plus_db, ...      %  6
            first_slp_plus_ang, ...     %  7
            first_slp_minus_db, ...     %  8
            first_slp_minus_ang, ...    %  9
            highest_slp_db, ...         % 10
            highest_slp_ang, ...        % 11
            highest2_slp_db, ...        % 12
            highest2_slp_ang, ...       % 13
            first_sln_plus_db, ...      % 14
            first_sln_plus_ang, ...     % 15
            first_sln_minus_db, ...     % 16
            first_sln_minus_ang, ...    % 17
            avg_sidelobe, ...           % 18
            percent_mainlobe, ...       % 19
```

```
            slp];                       % 20

end
costs=[data_bank(1,1), ...                  % 1 Number of Transmitters
        data_bank(1,2), ...                 % 2 Number of Receivers
        min(data_bank(:,3)), ...            % 3 Lowest Transmit level
        max(data_bank(:,4)), ...            % 4 Widest -3dB Beamwidth
        max(data_bank(:,5)), ...            % 5 Widest -20dB Beamwidth
        max(data_bank(:,10)), ...           % 6 Highest Sidelobe
        max(data_bank(:,12)), ...           % 7 2nd Highest Sidelobe
        mean(data_bank(:,18)), ...          % 8 Average sidelobe level
        min(data_bank(:,19)), ...           % 9 Min % energy in main lobe
        min(data_bank(:,20))];              % 10 Min peak sidelobe proximity

xdc_free(Th_xmit);
xdc_free(Th_rcv);
xdc_free(Single_Transmitter);
```

B.4 Cost Function Combination

```
function cost = costs_combiner2(cost_array, weights_array)
%  1: number of transmit elements
%  2: number of receive elements
%  3: minimum transmit SPL
%  4: maximum -3dB beamwidth
%  5: maximum -20dB beamwidth
%  6: highest sidelobe level
%  7: second highest sidelobe level
%  8: average sidelobe level
%  9: minimum % of energy in main lobe
% 10: minimum angular peak sidelobe proximity

w_cost_array=[];

cna1 = [ -2  -2 -195  0  0  80  80  80    0   0];  % these normalization
                                                    factors are designed to put the
cna2 = [198 198   55  3  5  80  80  80  100  45];  % individual cost
                                                    functions near the [0,1] interval
% 1: number of transmit elements
w_cost_array(1)=(cost_array(1)+cna1(1))/cna2(1)*weights_array(1);
% 2: number of receive elements
w_cost_array(2)=(cost_array(2)+cna1(2))/cna2(2)*weights_array(2);
% 3: minimum transmit SPL
w_cost_array(3)=(1-(cost_array(3)+cna1(3))/cna2(3))*weights_array(3);
% 4: maximum -3dB beamwidth
w_cost_array(4)=(cost_array(4)+cna1(4))/cna2(4)*weights_array(4);
% 5: maximum -20dB beamwidth
w_cost_array(5)=(cost_array(5)+cna1(5))/cna2(5)*weights_array(5);
% 6: highest sidelobe level
w_cost_array(6)=(cost_array(6)+cna1(6))/cna2(6)*weights_array(6);
% 7: second highest sidelobe level
w_cost_array(7)=(cost_array(7)+cna1(7))/cna2(7)*weights_array(7);
% 8: average sidelobe level
w_cost_array(8)=(cost_array(8)+cna1(8))/cna2(8)*weights_array(8);
% 9: minimum % of energy in main lobe
w_cost_array(9)=(1-(cost_array(9)+cna1(9))/cna2(9))*weights_array(9);
% 10: minimum angular peak sidelobe proximity
w_cost_array(10)=(1-(cost_array(10)+cna1(10))/cna2(10))*weights_array(10);

cost = sum(w_cost_array);
```

References

[1] G. Wade, "Acoustical Imaging: From Ancient Phoenicians to Modern Physicians," *Ultrasonics Symposium, Proceedings of the IEEE*, 1989, pp. 821-826.

[2] K. Erikson, A. Hairston, A. Nicoli, J. Stockwell, and T. White, *Ultrasonics Symposium, Proceedings of the IEEE*, Volume: 2 , 1997, pp. 1625 -1629.

[3] J. A. Jensen, "Linear Description of Ultrasound Imaging Systems, Notes for the International Summer School on Advanced Ultrasound Imaging." Technical University of Denmark. June 1999.

[4] J. F. Hopperstad and S. Holm, "Optimization of Sparse Arrays by an Improved Simulated Annealing Algorithm", *Proceedings of the International Workshop on Sampling Theory and Applications*, Loen, Norway, August 1999, pp. 91-95.

[5] J. A. Jensen, "Field: A Program for Simulating Ultrasound Systems." Medical Biol. Eng. Comp., *10th Nordic-Baltic Conference on Biomedical Imaging*, Vol. 4, Supplement 1, Part 1. 1996. pp. 351-353.

[6] S. I. Nikolov and J. A. Jensen, "Application of different spatial sampling patterns for sparse-array transducer design," *Proceedings of Ultrasonics International*, 1999. pp 921-922.

[7] P. Weber, A. Austeng, S. Holm, and N. Aakvaag, "1D and 2D sparse array optimization," *Instrumentation Science & Technology*, Vol. 27, No. 4, 1999, pp. 235-246.

[8] A. Trucco, "Synthesizing Wide-Band Arrays by Simulated Annealing", *Int. Conf. Oceans 2001 MTS/IEEE*, November 2001, pp. 989-994.

[9] K. Erikson, J. Stockwell, A. Hairston, J. Marciniec, and R. McPhie, "Matrix Arrays for the 21st Century", in *Acoustical Imaging*, Vol. 26, R. Maev, New York: Kluwer Academic/Plenum Publishers, 2002, pp. 363-372.